SPiiPlus ACSPL+

Command & Variable Reference Guide

Version NT 1.0

Version NT 1.0, 30 December 2010

COPYRIGHT

Copyright ® 1999 - 2010 ACS Motion Control Motion Control Ltd.

Changes are periodically made to the information in this document. Changes are published as release notes and are be incorporated into future revisions of this document.
No part of this document may be reproduced in any form without prior written permission from ACS Motion Control.

TRADEMARKS

Visual Basic and Windows are trademarks of Microsoft Corporation.
Any other companies and product names mentioned herein may be the trademarks of their respective owners.

Web Site: **www.AcsMotionControl.com**
Information: **info@AcsMotionControl.com**
Tech Support: **support@AcsMotionControl.com**

ACS Motion Control, Ltd.
Ramat Gabriel Industrial Park
POB 5668
Migdal HaEmek, 10500
ISRAEL
Tel: (972) (4) 6546440
Fax: (972) (4) 6546443

ACS Motion Control, Inc.
6575 City West Parkway
Eden Prairie, MN 55344
USA
Tel: 763-559-7669, 800-545-2980
Fax: 763-559-0110

ACS Motion Control (Korea)
Digital Empire Building D-191
980-3, Youngtong-dong, Youngtong-gu,
Suwon, Geonggi-do, 443-813, Korea
Tel: +82-31-202-3541
Fax: +82-31-202-3542

NOTICE

This guide contains proprietary information belonging to ACS Motion Control Ltd. Such information is supplied solely for the purpose of assisting users of the ACSPL+ motion control programing language
The text and graphics used in this guide are for the purpose of reference and illustration only. The specifications on which they are based are subject to change without notice.
The information in this document is deemed to be correct at the time of publishing. ACS Motion Control reserves the right to change specifications without notice. ACS Motion Control Motion Control is not responsible for incidental, consequential, or special damages of any kind in connection with using this document.

Changes in Version NT 1.0

Page	Change
38	Added description of the AXISDEF command.
39	Added ASSIGNPEG command
43	Added ASSIGNPOUTS command
45	Added ASSIGNFINS command
50	Added modified PEG_I command
51	Added modified PEG_R command
59	Added STOPPEG command
274	Added ECST variable
275	Added ECERR variable
304	Added description of the SYSINFO function.
317	Added ECIN function.
318	Added ECOUT function.
319	Added ECUNMAP function.
320	Added COREAD function
320	Added COWRITE function.
301	Modified GETCONF variable keys & definitions
305	Modified SETCONF variable keys & definitions
374	Communicationi Terminal commands updated to reflect the new SPiiPlus MMI Application Studio
417	Updated Error Codes chapter

Table of Contents

1 About This Document 1
1.1 Related Documents 1
1.2 Related SPiiPlus Tools 2
1.3 Conventions Used in this Guide 3
1.3.1 Text Conventions 3
1.3.2 Flagged Additional Information 3

2 ACSPL+ Commands 5
2.1 Axis Management Commands 7
2.1.1 BREAK 8
2.1.2 COMMUT 9
2.1.3 CONNECT 10
2.1.4 DEPENDS 13
2.1.5 DISABLE 13
2.1.6 ENABLE/ENABLEALL 16
2.1.7 FCLEAR 18
2.1.8 GO 19
2.1.9 GROUP 20
2.1.10 HALT 21
2.1.11 IMM 22
2.1.12 KILL 23
2.1.13 SAFETYCONF 24
2.1.14 SAFETYGROUP 26
2.1.15 SET 26
2.1.16 SPLIT 27
2.2 Interactive Commands 28
2.2.1 DISP 28
2.2.2 INTERRUPT 33
2.2.3 SEND 34
2.2.4 TRIGGER 35
2.3 Miscellaneous Commands 37
2.3.1 AXISDEF 38
2.3.2 ASSIGNPEG 39
2.3.3 ASSIGNPOUTS 43
2.3.4 ASSIGNFINS 45
2.3.5 DC 46
2.3.6 STOPDC 49
2.3.7 PEG_I 50
2.3.8 PEG_R 51
2.3.9 STARTPEG 54
2.3.10 READ 55
2.3.11 SETPROTECTION 56
2.3.12 SPDC 56
2.3.13 STOPPEG 59
2.3.14 WRITE 59
2.4 Motion Commands 60

2.4.1 ARC1 61
2.4.2 ARC2 62
2.4.3 JOG 63
2.4.4 LINE 64
2.4.5 MASTER 65
2.4.6 MPOINT 67
2.4.7 MPTP...ENDS 71
2.4.8 MSEG...ENDS 74
2.4.9 PATH...ENDS 77
2.4.10 POINT 78
2.4.11 PROJECTION 81
2.4.12 PTP 85
2.4.13 PVSPLINE...ENDS 86
2.4.14 SLAVE 89
2.4.15 STOPPER 90
2.4.16 TRACK 91
2.5 Program Flow Commands 92
2.5.1 Assignment Command 93
2.5.2 BLOCK...END 94
2.5.3 CALL 95
2.5.4 GOTO 96
2.5.5 IF, ELSEIF, ELSE...END 96
2.5.6 INPUT 98
2.5.7 LOOP...END 99
2.5.8 ON...RET 101
2.5.9 TILL 102
2.5.10 WAIT 103
2.5.11 WHILE...END 103
2.6 Program Management Commands 104
2.6.1 DISABLEON 105
2.6.2 ENABLEON 105
2.6.3 PAUSE 105
2.6.4 RESUME 106
2.6.5 START 106
2.6.6 STOP 107

3 ACSPL+ Variables 108
3.1 Axis Configuration Variables 114
3.1.1 AFLAGS 115
3.1.2 BOFFTIME 116
3.1.3 BONTIME 117
3.1.4 CERRA 118
3.1.5 CERRI 119
3.1.6 CERRV 120
3.1.7 DELI 121
3.1.8 DELV 122
3.1.9 EFAC 123
3.1.10 E2FAC 124

3.1.11 E_FREQ 125
3.1.12 E_TYPE 126
3.1.13 EOFFS 126
3.1.14 E2OFFS 127
3.1.15 E_SCMUL 128
3.1.16 ENTIME 129
3.1.17 ERRA 130
3.1.18 ERRI 131
3.1.19 ERRV 132
3.1.20 FVFIL 133
3.1.21 MFF 134
3.1.22 MFLAGS 135
3.1.23 RVFIL 139
3.1.24 SCCOFFS 140
3.1.25 SCSOFFS 141
3.1.26 SETTLE 141
3.1.27 SLLIMIT 142
3.1.28 SLPMAX 143
3.1.29 SLPMIN 144
3.1.30 SRLIMIT 145
3.1.31 STEPF 146
3.1.32 STEPW 147
3.1.33 SYNV 147
3.1.34 TARGRAD 149
3.1.35 XACC 150
3.1.36 XCURI 150
3.1.37 XCURV 151
3.1.38 XRMS 152
3.1.39 XRMST 153
3.1.40 XSACC 154
3.1.41 XVEL 155
3.2 Axis State Variables 156
3.2.1 APOS 156
3.2.2 AST 157
3.2.3 DAPOS 158
3.2.4 F2POS 159
3.2.5 FACC 159
3.2.6 FPOS 160
3.2.7 FVEL 160
3.2.8 F2VEL 161
3.2.9 IND 161
3.2.10 IST 162
3.2.11 M2ARK 163
3.2.12 MARK 164
3.2.13 MPOS 164
3.2.14 MST 165
3.2.15 PE 166
3.2.16 RACC 166

3.2.17 ROFFS 167
3.2.18 RPOS 168
3.2.19 RVEL 169
3.3 Data Collection Variables 169
3.3.1 DCN 169
3.3.2 DCP 170
3.3.3 S_DCN 171
3.3.4 S_DCP 172
3.3.5 S_ST 172
3.4 Input and Output Variables 173
3.4.1 AIN 174
3.4.2 AOUT 174
3.4.3 EXTIN 175
3.4.4 EXTOUT 176
3.4.5 IN 176
3.4.6 OUT 177
3.5 Monitoring Variables 178
3.5.1 SMNA 178
3.5.2 SMNV 179
3.5.3 TIME 179
3.5.4 USAGE 180
3.6 Motion Variables 180
3.6.1 ACC 181
3.6.2 DEC 182
3.6.3 GACC 183
3.6.4 GJERK 183
3.6.5 GMOT 184
3.6.6 GMQU 185
3.6.7 GMTYPE 185
3.6.8 GPATH 186
3.6.9 GPHASE 187
3.6.10 GRTIME 188
3.6.11 GSEG 188
3.6.12 GSFREE 189
3.6.13 GVEC 190
3.6.14 GVEL 190
3.6.15 JERK 191
3.6.16 KDEC 192
3.6.17 NVEL 193
3.6.18 TPOS 194
3.6.19 VEL 195
3.6.20 VELBRK 196
3.7 Program Execution Control Variables 197
3.7.1 ONRATE 197
3.7.2 PCHARS 198
3.7.3 PERL 198
3.7.4 PERR 199
3.7.5 PEXL 199

3.7.6 PFLAGS 200
3.7.7 PLINES 201
3.7.8 PRATE 201
3.7.9 PST 202
3.8 Safety Control Variables 203
3.8.1 AERR 204
3.8.2 FAULT 204
3.8.3 FDEF 207
3.8.4 FMASK 210
3.8.5 MERR 213
3.8.6 SAFIN 215
3.8.7 SAFINI 216
3.8.8 S_ERR 217
3.8.9 S_FAULT 217
3.8.10 S_FDEF 220
3.8.11 S_FMASK 221
3.8.12 S_SAFIN 223
3.8.13 S_SAFINI 224
3.9 Induction Motor Variables 225
3.9.1 SLCFIELD 225
3.9.2 SLCSLIP 226
3.10 Nanomotion Variables 228
3.10.0.1 SLDZMIN 228
3.10.0.2 SLDZMAX 229
3.10.0.3 SLZFF 230
3.11 Servo-Loop Variables 231
3.11.1 DCOM 231
3.11.2 Servo-Loop Current Variables 232
3.11.2.1 SLBIASA 233
3.11.2.2 SLBIASB 233
3.11.2.3 SLIKI 234
3.11.2.4 SLIKP 235
3.11.3 Servo-Loop Velocity Variables 236
3.11.3.1 SLVKI 236
3.11.3.2 SLVKP 237
3.11.3.3 SLVLI 237
3.11.4 Servo-Loop Velocity Notch Filter Variables 238
3.11.4.1 SLVNFRQ 238
3.11.4.2 SLVNWID 239
3.11.4.3 SLVNATT 239
3.11.5 Servo-Loop Velocity Low Pass Filter Variables 240
3.11.5.1 SLVSOF 240
3.11.5.2 SLVSOFD 241
3.11.6 Servo-Loop Velocity Bi-Quad Filter Variables 242
3.11.6.1 SLVB0DD 242
3.11.6.2 SLVB0DF 243
3.11.6.3 SLVB0ND 243
3.11.6.4 SLVB0NF 244

3.11.7 Servo-Loop Position Variables 245
3.11.7.1 SLDRA 245
3.11.7.2 SLDRX 246
3.11.7.3 SLPKP 247
3.11.8 Servo-Loop Commutation Variables 248
3.11.8.1 SLCHALL 248
3.11.8.2 SLCNP 249
3.11.8.3 SLCPA 249
3.11.8.4 SLCOFFS 250
3.11.8.5 SLCORG 251
3.11.8.6 SLCPRD 251
3.11.9 Servo-Loop Compensations Variables 252
3.11.9.1 SLAFF 252
3.11.9.2 SLFRC 253
3.11.9.3 SLFRCD 253
3.11.10 Servo-Loop Miscellaneous Variables 254
3.11.10.1 SLABITS 255
3.11.10.2 SLPBITS 256
3.11.10.3 SLIOFFS 256
3.11.10.4 Feedback Routing 257
3.11.10.5 SLVRAT 261
3.12 System Configuration Variables 262
3.12.1 CFG 262
3.12.2 CTIME 263
3.12.3 IENA 263
3.12.4 IMASK 265
3.12.5 ISENA 266
3.12.6 S_FLAGS 267
3.13 Communication Variables 268
3.13.1 BAUD 269
3.13.2 COMMCH 270
3.13.3 COMMFL 271
3.13.4 CONID 272
3.13.5 ECHO 273
3.13.6 ECST 274
3.13.7 ECERR 275
3.13.8 DISPCH 276
3.13.9 TCPIP 277
3.13.10 TCPIP2 278
3.13.11 TCPPORT 279
3.13.12 UDPPORT 280
3.14 Miscellaneous 281
3.14.1 FK 281
3.14.2 XARRSIZE 282

4 ACSPL+ Functions 283
4.1 Arithmetical Functions 285
4.1.1 ABS 286

4.1.2 ACOS 286
4.1.3 ASIN 286
4.1.4 ATAN 287
4.1.5 ATAN2 288
4.1.6 CEIL 288
4.1.7 COS 289
4.1.8 EXP 289
4.1.9 FLOOR 290
4.1.10 HYPOT 290
4.1.11 LDEXP 291
4.1.12 LOG 291
4.1.13 LOG10 292
4.1.14 POW 292
4.1.15 SIGN 293
4.1.16 SIN 293
4.1.17 SQRT 294
4.1.18 TAN 294
4.2 Array Processing Functions 295
4.2.1 AVG 295
4.2.2 COPY 295
4.2.3 FILL 297
4.2.4 MAX 298
4.2.5 MAXI 299
4.2.6 MIN 299
4.2.7 MINI 300
4.3 Miscellaneous Functions 301
4.3.1 GETCONF 301
4.3.2 SYSINFO 304
4.3.3 GETVAR 305
4.3.4 SETCONF 305
4.3.5 SETVAR 312
4.3.6 STR 312
4.3.7 STRTONUM 313
4.3.8 NUMTOSTR 314
4.3.9 BCOPY 315
4.3.10 ECIN 317
4.3.11 ECOUT 318
4.3.12 ECUNMAP 319
4.4 CoE Functions 319
4.4.1 COEREAD 320
4.4.2 COEWRITE 320
4.5 Servo Processor Functions 321
4.5.1 GETSP 322
4.5.2 GETSPA 323
4.5.3 GETSPV 323
4.5.4 MONSP 323
4.5.5 MONSPV 325
4.5.6 SETSP 326

4.5.7 SETSPV 327
4.6 Signal Processing Functions 327
4.6.1 DEADZONE 328
4.6.2 DSIGN 331
4.6.3 DSTR 332
4.6.4 EDGE 334
4.6.5 INP 335
4.6.6 INTGR 337
4.6.7 LAG 338
4.6.8 MAP 340
4.6.9 MAPB 342
4.6.10 MAPN 344
4.6.11 MAPNB 346
4.6.12 MAPNS 347
4.6.13 MAPS 349
4.6.14 MAP2 351
4.6.15 MAP2B 354
4.6.16 MAP2N 356
4.6.17 MAP2NB 359
4.6.18 MAP2NS 361
4.6.19 MAP2S 364
4.6.20 MATCH 366
4.6.21 OUTP 368
4.6.22 RAND 370
4.6.23 ROLL 371
4.6.24 SAT 372

5 Terminal Commands 374
5.1 Entering Terminal Commands 374
5.2 Query Commands 375
5.2.1 Default Query Formats 376
5.2.2 Predefined Query Output Formats 377
5.2.3 User-Defined Query Output Format 378
5.3 Program Management Commands 378
5.3.1 Program Management Command Arguments 378
5.3.2 Program Buffer Commands 380
5.3.2.1 Open/Close Buffer (#) 380
5.3.2.2 D 382
5.3.2.3 F/FI 383
5.3.2.4 L 384
5.3.2.5 PROTECT 386
5.3.2.6 UNPROTECT 386
5.3.3 RESET 387
5.3.4 Listing Program Variables 387
5.3.4.1 VGR 387
5.3.4.2 VSD 388
5.3.4.3 VS/VSG 388
5.3.4.4 VSF/VSGF 390

5.3.4.5 VG/VGF 390
5.3.4.6 VL/VLF 391
5.3.4.7 V/VF 391
5.3.4.8 VSP 392
5.3.4.9 VST/VSGT 393
5.3.4.10 VSTF/VSGTF/VSDT 393
5.3.4.11 VGV 394
5.3.5 Program Handling Commands 394
5.3.5.1 C 395
5.3.5.2 X 396
5.3.5.3 S/SR 397
5.3.5.4 P 398
5.3.6 Debug Commands 399
5.3.6.1 XS 399
5.3.6.2 XD 400
5.3.6.3 BS 400
5.3.6.4 BR 401
5.4 System Commands 402
5.4.1 SI 402
5.4.2 IR 407
5.4.3 U 409
5.4.4 TD 409
5.4.5 SC 409
5.4.6 ETHERCAT 411
5.4.7 CC 416

6 SPiiPlus Error Codes 417
6.1 ACSPL+ Syntax Errors 417
6.2 ACSPL+ Compilation Errors 434
6.3 ACSPL+ Runtime Errors 442
6.4 Motion Termination Errors 454
6.5 System Errors 459
6.6 EtherCAT Errors 463

List of Figures

Figure 1 CONNECT Using MAP Function 12
Figure 2 DISABLE and Mechanical Brake Output Process- Positive BONTIME 15
Figure 3 DISABLE and Mechanical Brake Output Process - Negative BONTIME 15
Figure 4 ENABLE and Mechanical Brake Output Process - Positive BOFFTIME 17
Figure 5 ENABLE and Mechanical Brake Output Process - Negative BOFFTIME . . . 17
Figure 6 ARC1 Coordinate Specification 61
Figure 7 ARC2 Center Point and Rotation Angle Specification 62
Figure 8 SLAVE /pt Illustration 67
Figure 9 Single-Axis Motion Using MPTP 73
Figure 10 Two-Axis Group Motion Using MPTP/v 74
Figure 11 Results of Example MSEG 76
Figure 12 Like PVSPLINE Drawing 78
Figure 13 PROJECTION of the XA Plane 83
Figure 14 FPOS - PROJECTION Example 83
Figure 15 PROJECTION Example - Final Result 84
Figure 16 PVSPLINE Motion 88
Figure 17 Use of STOPPER 91
Figure 18 Illustration of COPY Function 297
Figure 19 Illustration of Shift 324
Figure 20 Symmetrical Dead Zone Example 330
Figure 21 Asymmetrical Dead Zone Example 331
Figure 22 DSIGN Function Example 332
Figure 23 EDGE Function Example 335
Figure 24 INTGR Function Example 338
Figure 25 LAG Function Example 339
Figure 26 MAP Example on the Scope 341
Figure 27 MAPB Example on the Scope 343
Figure 28 MAPN Example on the Scope 345
Figure 29 MAPNB Example on the Scope 347
Figure 30 MAPNS Example on the Scope 349
Figure 31 MAPS Example on the Scope 351
Figure 32 MAP2 Example on the Scope 353
Figure 33 MAP2B Example on the Scope 356
Figure 34 MAP2N Example on the Scope 358
Figure 35 MAP2NB Example on the Scope 361
Figure 36 MAP2NS Example on the Scope 363
Figure 37 MAP2S Example on the Scope 366
Figure 38 ROLL Example on the Scope 372
Figure 39 SAT Example on the Scope 373
Figure 40 Communication Terminal Window 374
Figure 41 Interaction of Program Buffer States 394

List of Tables

Table 1 Related Documentation 1
Table 2 Related SPiiPlus Tools 2
Table 3 ACSPL+ Command Set. 5
Table 4 DISP Command Option Escape Sequences 29
Table 5 Type Characters. 30
Table 6 Channel Designation for TRIGGER 36
Table 7 Mapping PEG Engines to Encoders (Servo Processor 0) 40
Table 8 Mapping PEG Engines to Encoders (Servo Processor 1) 40
Table 9 Digital Outputs Assignment for Use as PEG State and PEG Pulse Outputs (Servo Processor 0) 41
Table 10 Digital Outputs Assignment for Use as PEG State and PEG Pulse Outputs (Servo Processor 1) 42
Table 11 Mapping of Engine Outputs to Physical Outputs (Servo Processor 0) 43
Table 12 Mapping of Engine Outputs to Physical Outputs (Servo Processor 1) 44
Table 13 Mapping of Engine Inputs to Physical Inputs (Servo Processor 0). 45
Table 14 Mapping of Engine Inputs to Physical Inputs (Servo Processor 1). 45
Table 15 PEG Output Signal Configuration. 52
Table 16 Common Monitored SPDC Variables. 57
Table 17 Matrix Values 82
Table 18 IF Control Structures. 97
Table 19 Alphabetical Listing of All ACSPL+ Variables 108
Table 20 AFLAGS Bit Description 115
Table 21 MFLAGS Bit Designators. 135
Table 22 AST Bit Descriptions 157
Table 23 MST Bit Descriptions 165
Table 24 PFLAGS Bit Description. 200
Table 25 PST Bit Description. 202
Table 26 Axis Fault Bits. 204
Table 27 FDEF Bit Description 207
Table 28 FMASK Bit Description 211
Table 29 MERR Return Values 213
Table 30 SAFINI Valid Bits. 216
Table 31 S_FAULT Fault Bits 217
Table 32 S_FDEF Bit Description 220
Table 33 S_FMASK Bit Description 222
Table 34 SLCROUT Values. 258
Table 35 SLPROUT Values 259
Table 36 SLVROUT Values. 260
Table 37 IENA Bit Description 264
Table 38 ISENA Bit Description 266
Table 39 S_FLAGS Bit Description. 267
Table 40 COMMCH Values. 270
Table 41 COMMFL Bit Descriptions. 271
Table 42 ECHO Channel Numbers 273
Table 43 ECST Bits 274

Table 44 EtherCAT Error Codes 275
Table 45 DISPCH Channel Numbers. 276
Table 46 GETCONF Return Values. 302
Table 47 SYSINFO Return Values 304
Table 48 16-bit Binary value Template 306
Table 49 SETCONF Arguments. 306
Table 50 MAP Array 341
Table 51 MAPB Array 343
Table 52 MAPN Array 345
Table 53 MAPNB Array. 347
Table 54 MAPNS Array 349
Table 55 MAPS Array 350
Table 56 MAP2 Table 353
Table 57 MAP2B Table 355
Table 58 MAP2N Table 358
Table 59 MAP2NB Table. 360
Table 60 MAP2NS Table 363
Table 61 MAP2S Table 365
Table 62 Line Designation 379
Table 63 ACSPL+ Syntax Errors 417
Table 64 ACSPL+ Compilation Errors 434
Table 65 ACSPL+ Runtime Errors. 442
Table 66 ACSPL+ Motion Termination Errors 454
Table 67 ACSPL+ System Errors. 459
Table 68 ACSPL+ EtherCAT Errors 463

1 About This Document

The SPiiPlus Command & Variable Reference Guide details all of the elements making up the SPiiPlus ACSPL+ Programming Language as well as the command set that may be entered through the SPiiPlus MMI Terminal.

The document is organized as follows:

- **Chapter 2 - ACSPL+ Commands**
- **Chapter 3 - ACSPL+ Variables**
- **Chapter 4 - ACSPL+ Functions**
- **Chapter 5 - Terminal Commands**
- **Chapter 6 - SPiiPlus Error Codes**

1.1 Related Documents

The following documents may be of assistance as additional references:

Table 1 Related Documentation

Document	Description
SPiiPlus C Library Reference	C++ and Visual Basic® libraries for host PC applications. This guide is applicable for all the SPiiPlus motion control products.
SPiiPlus COM Library Reference	COM Methods, Properties, and Events for Communication with the Controller.
SPiiPlus MMI Application Studio User Guide	A complete guide for using the SPiiPlus MMI Application Studio and associated monitoring tools.
SPiiPlus Utilities User Guide	A guide for using the SPiiPlus User Mode Driver (UMD) for settingup communication with the SPiiPlus motion controller.
SPiiPlus NT/DC Hardware Guide	Technical description of the SPiiPlus NT/DC product line.
SPiiPlus PDMnt Hardware Guide	Technical description of the SPiiPlus PDMnt Network Interface.
SPiiPlus SDMnt Hardware Guide	Technical description of the SPiiPlus SDMnt Step Motor Drive Module.
SPiiPlus UDMnt Hardware Guide	Technical description of the SPiiPlus UDMnt Universal Drive Module.
MC4U-CS Control Module Hardware Guide	Technical description of the MC4U Control Module integrated motion control product line.
HSSI Expansion Modules Guide	High-Speed Synchronous Serial Interface (HSSI) for expanded I/O, distributed axes, and nonstandard devices.
SPiiPlus NT PEG Commands Application Notes	Provides details on the PEG command set that is implemented for the SPiiPlus NT/DC product line.

1.2 Related SPiiPlus Tools

The following software tools are included with all SPiiPlus motion controllers:

Table 2 Related SPiiPlus Tools

Tool	Description
SPiiPlus MMI Application Studio	A multipurpose user interface with the controller including: program management, motion management, Communication Terminal, four-channel digital oscilloscope, Safety and I/O signals monitor, signal tuning and adjustment, and a fully interactive simulator.
SPiiPlus C Library	A DLL (Dynamic Link Library) that supports host application programming in a variety of languages including C/C++. The library introduces a new level of application support with a built-in controller simulator and it also provides outstanding debugging capabilities. All tools are provided with a full simulator of the controller.
SPiiPlus COM Library	A DLL (Dynamic Link Library) that supports host application programming in a variety of languages including Visual Basic, LabView, and more. The library introduces a new level of application support with a built-in controller simulator and it also provides outstanding debugging capabilities. All tools are provided with a full simulator of the controller.

1.3 Conventions Used in this Guide

1.3.1 Text Conventions

Several text formats and fonts are used in the text to convey information about the text. Command syntax follows normal usage.

Text	Description
BOLD CAPS	ACSPL+ elements (commands, functions, operators, standard variables, etc.) when mentioned in the text. Software tool menus, menu items, dialog box names and dialog box elements.
bold	Emphasis or an introduction to a key concept.
`Monospace`	Code examples.
`Italic monospace`	Information in code examples that the user provides.
ALL CAPS	(Keyboard) key names [example: SHIFT key].
Bold Blue	Links within this document, to web pages, and to e-mail addresses.
|	Used in command syntax, separates mutually exclusive choices, only one of which is to be selected.
{ }	Used in command syntax to indicate set of choices from which one is to be selected.
[ ]	Used in command syntax to indicate optional input items.

1.3.2 Flagged Additional Information

The following symbols are used to flag additional information:

Note	*Notes include helpful information or tips.*

Caution	***A Caution describes a condition that may result in damage to equipment.***

Warning	***A Warning describes a condition that may result in serious bodily injury or death.***

Advanced	*Indicates a topic for advanced users.*

Model	*Highlights a specification, procedure, condition, or statement that depends on the product model.*

2 ACSPL+ Commands

ACSPL+ comes with a complete programming command set. The commands are divided into following categories:

- **Axis Management Commands**
- **Interactive Commands**
- **Miscellaneous Commands**
- **Motion Commands**
- **Program Flow Commands**
- **Program Management Commands**

This chapter covers the ACSPL+ programming commands.

The ACSPL+ command set is:

Table 3 ACSPL+ Command Set (page 1 of 3)

Command	Action
ARC1	Adds a circle segment to **MSEG...ENDS** motion.
ARC2	Adds a circle segment to **MSEG...ENDS** motion.
ASSIGNFINS	Assigns PEG inputs
ASSIGNPEG	Assigns PEG engine to encoder
ASSIGNPOUTS	Assigns PEG outputs
AXISDEF	Assigns an alias to an axis
BLOCK...END	Performs a group of ACSPL+ commands in one controller cycle.
BREAK	Immediately terminates a motion and provides smooth transition to the next motion in the motion queue.
CALL	Calls subroutine.
COMMUT	Performs auto commutation for DC brushless (AC servo) motors.
CONNECT	Defines a formula for calculating reference position (**RPOS**).
DC	Activates data collection.
DEPENDS	Specifies a logical dependence between a motor and axes.
DISABLE	Shuts off one or more drives. **DISABLE ALL** provides **DISABLE** operation for all axes.
DISABLEON	Disables autoroutine activation in a buffer.
DISP	Builds a string and sends it to the default communication channel.
ENABLE/ENABLEALL	Activates one or more drives
ENABLEON	Enables autoroutine activation in a buffer.
FCLEAR	Clears faults.
GO	Starts a motion that was created using the **/w** command option.
GOTO	Transfers program execution to another point in the program.

Table 3 ACSPL+ Command Set (page 2 of 3)

Command	Action
GROUP	Defines an axis-group for coordinate multi-axis motion.
HALT	Terminates one or more motions using a third-order deceleration profile (**DEC** deceleration). **HALTALL** provides **HALT** operation for all axes.
IF, ELSEIF, ELSE...END	**IF** command structure.
IMM	Provides on-the-fly change of motion parameters.
INPUT	Suspends program execution pending user input
INTERRUPT	Causes an interrupt that can be intercepted by the host.
JOG	Creates a jog motion.
KILL	Terminates one or more motions using a second-order deceleration profile and the **KDEC** deceleration value. **KILLALL** provides **KILL** operation for all axes.
LINE	Adds a linear segment to **MSEG...ENDS** motion.
LOOP...END	Loop command structure.
MASTER	Defines a formula for calculating **MPOS**.
MPOINT	Adds a set of points to **MPTP...ENDS**, **PATH...ENDS** **or** **PVSPLINE...ENDS** motion.
MPTP...ENDS	Creates a multipoint motion.
MSEG...ENDS	Creates a segmented motion.
ON...RET	The autoroutine structure. An the autoroutine is automatically executed when a specific condition is satisfied. The routine interrupts the currently executing program, executes the commands specified in the autoroutine body, and then returns control to the interrupted program.
PATH...ENDS	Creates an arbitrary path motion with linear interpolation between the specified points.
PAUSE	Suspends program execution in a buffer.
PEG_I	Activates incremental PEG, where pulses are generated at predefined, fixed position intervals.
PEG_R	Activates random, table-based PEG, where pulses are generated at predefined positions.
POINT	Adds a point to **MPTP...ENDS**, **PATH...ENDS,** or **PVSPLINE...ENDS** motion.
PROJECTION	An expansion command to the **MSEG...ENDS** set of commands, that allows the controller to perform a three dimensional segmented motion such as creating arcs and lines on a user-defined plane.
PTP	Creates a point-to-point motion.
PVSPLINE...ENDS	Creates an arbitrary path motion with spline interpolation between the specified points.
READ	Reads an array from a file in the flash memory.
RESUME	Resumes program execution in a buffer.
SAFETYCONF	Configures fault processing for one or more axes.

Table 3 ACSPL+ Command Set (page 3 of 3)

Command	Action
SAFETYGROUP	Activates the fault response for all axes in the axis_list when any axis triggers the fault, and manages the axes as a block in response to **KILL** and **DISABLE**.
SEND	Same as **DISP**, but also specifies the communication channel or channels.
SET	Defines the current value of either feedback (**FPOS**), reference (**RPOS**), or axis (**APOS**) position.
SETPROTECTION	Changes protection attribute for a variable.
SLAVE	Creates a master-slave motion.
SPDC	Activates data collection from a SP variable.
SPLIT	**SPLIT** breaks apart an axis group. **SPLITALL** breaks apart all axis groups. See **GROUP**.
START	Activates program execution in a buffer.
STARTPEG	Activates PEG motion for specified axis.
STOP	Terminates program execution in a buffer.
STOPDC	Terminates data collection.
STOPPEG	Terminates PEG operation.
STOPPER	Adds a segment separator to **MSEG...ENDS** motion.
TILL	Delays program execution until a specified expression produces a non-zero (true) result.
TRACK	Creates tracking motion.
TRIGGER	Specifies a triggering condition. Once the condition is satisfied, the controller issues an interrupt to the host computer.
WAIT	Delays program execution for a specified number of milliseconds.
WHILE...END	While command structure.
WRITE	Writes an array to a file in the flash memory.

2.1 Axis Management Commands

The Axis Management commands are:

Command	Action
BREAK	Immediately terminates a motion and provides smooth transition to the next motion in the motion queue.
COMMUT	Performs auto commutation for DC brushless (AC servo) motors.
CONNECT	Defines a formula for calculating reference position (**RPOS**).
DEPENDS	Specifies a logical dependence between a motor and axes.
DISABLE	Shuts off one or more drives. **DISABLE ALL** provides **DISABLE** operation for all axes.
ENABLE/ENABLEALL	Activates one or more drives

Command	Action
FCLEAR	Clears faults.
GO	Starts a motion that was created using the **/w** command option.
GROUP	Defines an axis-group for coordinate multi-axis motion.
HALT	Terminates one or more motions using a third-order deceleration profile (**DEC** deceleration). **HALTALL** provides **HALT** operation for all axes.
IMM	Provides on the fly change of motion parameters.
KILL	Terminates one or more motions using a second-order deceleration profile and the **KDEC** deceleration value. **KILLALL** provides **KILL** operation for all axes.
SAFETYCONF	Configures fault processing for one or more axes.
SAFETYGROUP	Creates a safety axis group. When any axis in the group triggers a fault, the fault affects all axes in the group.
SET	Defines the current value of either feedback (**FPOS**), reference (**RPOS**), or axis (**APOS**) position.
SPLIT	**SPLIT** breaks apart an axis group - see **GROUP**. **SPLITALL** breaks apart all axis groups.

2.1.1 BREAK

Description

BREAK immediately terminates the currently executed motion of the specified axis without building a deceleration profile, and initiates the next motion in the axis motion queue, if it exists.

Syntax

BREAK *axis_list*

Arguments

axis_list	Axis or list of axes, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.

Comments

BREAK executes differently in the following cases:

1. When the next motion waits in the motion queue, **BREAK** terminates the current motion and starts the next motion immediately.
2. When there is no next motion in the motion queue **BREAK** has no immediate effect. The current motion continues until the next motion appears in the motion queue. At that moment the controller breaks the current motion and provides a smooth velocity transition profile from motion to motion. If the current motion finishes before the next motion comes to the queue, the command has no effect.

COM Library Methods
Break

C Library Functions
acsc_Break

Example:

```
PTP 0, 1E8              !Move to point 1E8
PTP 0, -1E8             !Add another motion to the motion queue
WAIT 10000              !Wait 10 seconds
BREAK 0                 !Terminate the first motion (PTP 0, 1E8)
                        !and immediately start the next motion:
                        !(PTP 0, -1E8)
STOP                    !End program
```

2.1.2 COMMUT

Description
COMMUT performs auto commutation and may be used when the following conditions hold true:

- ❑ motor is DC brushless (AC servo)
- ❑ motor is enabled
- ❑ motor is idle
- ❑ The axis is already configured and properly tuned

COMMUT is used in auto commutation-based startup programs.

Syntax
COMMUT *axis [,excitation_current][,settle_time][,slope]*

Arguments

axis	The affected axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
excitation_current	Optional - Given as a percentage of the full current command. The default value is set to 98% of **XRMS**.

settle_time	Optional - Determines settling time for the auto commutation process initiated by **COMMUT**. The default value is 500 msec. The entire auto commutation process lasts approximately three times longer, since the command executes the algorithm three times for verification.
slope	Optional - The slope parameter is optional, and used only in special cases. If a value is assigned to this parameter then the excitation current command builds up with some slope. The parameter sets the duration of the current build-up process in milliseconds. It is usually recommended to omit this parameter, in which case the excitation current is built instantly.

Comments

The excitation current, settle time and slope are optional parameters for the auto commutation process initiated by **COMMUT**.

Refer to the relevant section in the Setup Guide for a complete description of the commutation process.

Example:

```
COMMUT 0,80,100,30      !Commut 0 axis with an excitation current
                        !of 80%. Settling time is 100msec, and a
                        !current build-up slope of 30msec.
```

2.1.3 CONNECT

Description

CONNECT defines a formula for calculating reference position (**RPOS**). This formula can include any other axes variables. **DEPENDS** must follow **CONNECT**.

Syntax

CONNECT *axis_RPOS = formula*

Comments

1. The default relation between an axis position (**APOS**) and its reference (motor) position (**RPOS**) is 1:1.
2. Defining a different relation can be very useful for mechanical error corrections, dynamic error compensation, backlash compensation, inverse kinematics and more.
3. If the **CONNECT** relation is based on an other axis position, it creates a strict link (like a mechanical connection) between all defined axes for as long as the function is active.
4. Variable **MFLAGS**<*axis*>.17 (bit 17) disables or enables a customized (non-default) **CONNECT** formula definition. See **MFLAGS**.
5. After **CONNECT** it is recommended to initialize **ROFFS** with the first value in the correction table (as seen in the example:

   SET RPOS0=MAP(APOS0,ARRAY,100,200))

   This forces **ROFFS** to be zero and prevents the creation of a constant offset to **RPOS**.

6. **ENABLE/ENABLEALL**, **DISABLE** and **KILL** change the value of **ROFFS**. Therefore, if these commands follow **CONNECT**, then redefine the **CONNECT** formula, and **RPOS** should be initialized to nearest value.
7. To stop motion after using **CONNECT**, use **HALT** instead of **KILL**. **HALT** does not affect the **ROFFS** variable.

Related ACSPL+ Commands
DEPENDS

Related ACSPL+ Variables
RPOS, **APOS**, **ROFFS**

Example

```
GLOBAL REAL ARRAY(6)     !Define ARRAY.
ARRAY(0)=60;ARRAY(1)=40;ARRAY(2)=90;ARRAY(3)=-40;ARRAY(4)=60; ARRAY(5)=10
                         !Populate Correction point ARRAY for MAP
                         !function.
MFLAGS(0).17=1           !Set default connection between APOS and
                         !RPOS (RPOS=APOS).
ENABLE 0                 !Enable 0 axis.
MFLAGS(0).17=0           !Set non-default connection between APOS
                         !and RPOS (RPOS is a function of APOS).
CONNECT RPOS0=APOS0+MAP(APOS0,ARRAY,100,200)
                         !CONNECT formula between RPOS0 and
                         !APOS0 using the MAP function with
                         !correction table ARRAY.
DEPENDS 0,0              !Assign Axis 0 to Motor 0. See DEPENDS.
SET APOS0=0;SET RPOS0=0  !Initialize APOS and RPOS at 0.
PTP 0, MAP(APOS0,ARRAY,100,200)
                         !Moves axis 0 to the first point in the
                         !correction ARRAY to avoid a constant
                         !offset in ROFFS, as explained in Comment 2.
PTP 0, 1300              !Move to 1300. Each point during the motion
                         !is modified according to the correction
                         !ARRAY in the MAP function.
STOP                     !End program
```

Figure 1 illustrates the results of this example on the SPiiPlus MMI Application Studio Scope.

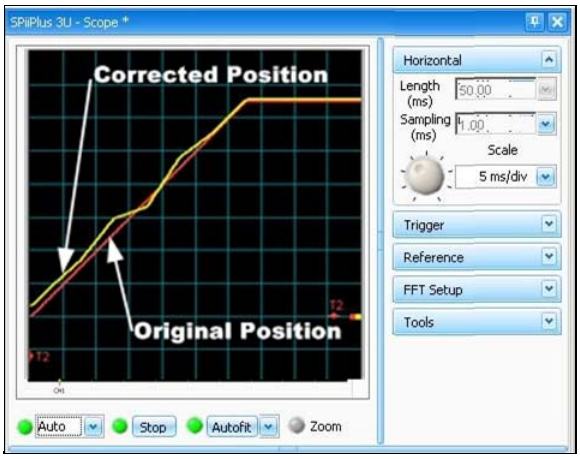


Figure 1 CONNECT Using MAP Function

2.1.4 DEPENDS

Description

DEPENDS is used only following **CONNECT**.

DEPENDS specifies a logical dependence between a physical axis (motor) and the same or other logical axes. By default, the physical axis (motor) is assigned to its axis. **DEPENDS** is necessary because the controller is generally not capable of deriving dependence information from the **CONNECT** formula.

Syntax

DEPENDS *physical_axis, axis_list*

Arguments

physical_axis	Physical axis (motor).
axis_list	Axis or list of axes, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.

Comments

1. Once a **CONNECT** command is executed, the controller resets the motor dependence information to the default-the motor depends only on the corresponding axis.
2. If a connection formula actually causes the motor to be dependent on another axis / axes, the **DEPENDS** command must follow to specify actual dependence.

Related ACSPL+ Commands

CONNECT

Example

See the examples from **CONNECT**.

2.1.5 DISABLE

Description

DISABLE deactivates one, several, or using **DISABLEALL**, all drives. After **DISABLE,** **RPOS**= **FPOS** which means that no position error exists, or **PE**<*axis*> = 0.

Syntax

DISABLE *axis_list [,reason]*

Arguments

axis_list	Axis, or list of axes, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
reason	***reason*** is an optional parameter. ***reason*** must be an integer constant or expression that equates to an integer and specifies a reason why the motor was disabled. If the parameter is specified, its value is stored in the **MERR** variable. If the parameter is omitted, **MERR** stores zero after the disable operation

Related ACSPL+ Commands
ENABLE/ENABLEALL, **FCLEAR**, **DISABLEALL**

Related ACSPL+ Variables
MERR, **FPOS**, **RPOS**

COM Library Methods
Disable, DisableM

C Library Functions
acsc_Disable, acsc_DisableM

Examples:
Figure 4 and **Figure 5** illustrate the **DISABLE** process using positive and negative **BONTIME** values.

The examples are followed by screens illustrating the results of DISABLE for both positive and negative BONTIME.

Example 1:

```
DISABLE (0,1)          !Disables the 0 and 1 motors. A fault
                       !notification window will be displayed.
```

Example 2:

```
DISABLE (0,1), 6100    !Disable 0 and 1 motors, store 6100 as the
                       !reason. 6100 specifies a user-defined error code
                       !which is stored in MERR.
```

Example 3:

```
DISABLE ALL            !Disable all motors.
```

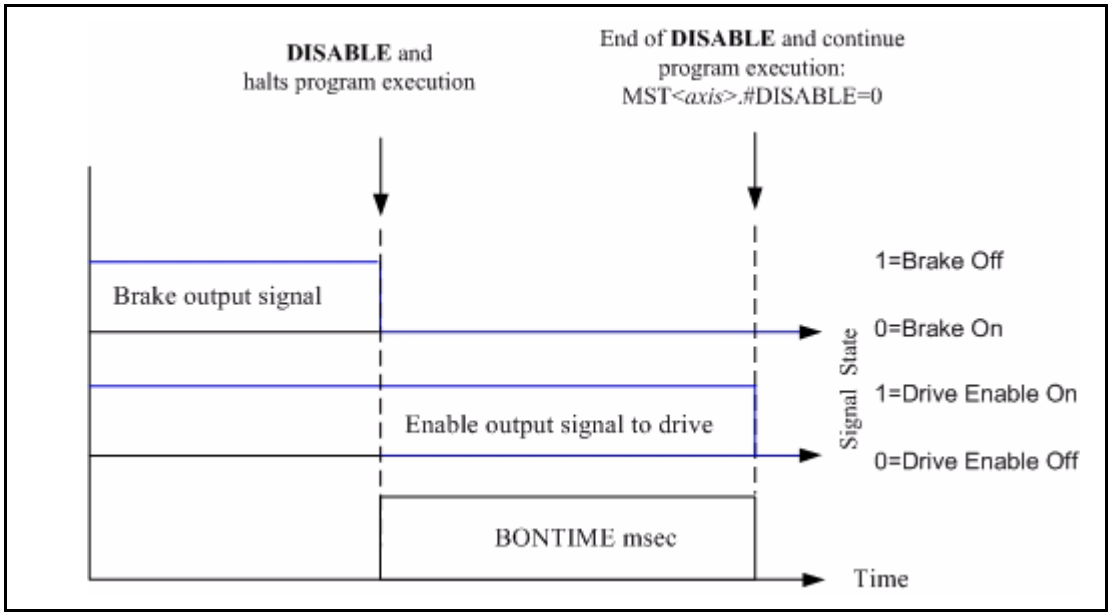


Figure 2 DISABLE and Mechanical Brake Output Process- Positive BONTIME

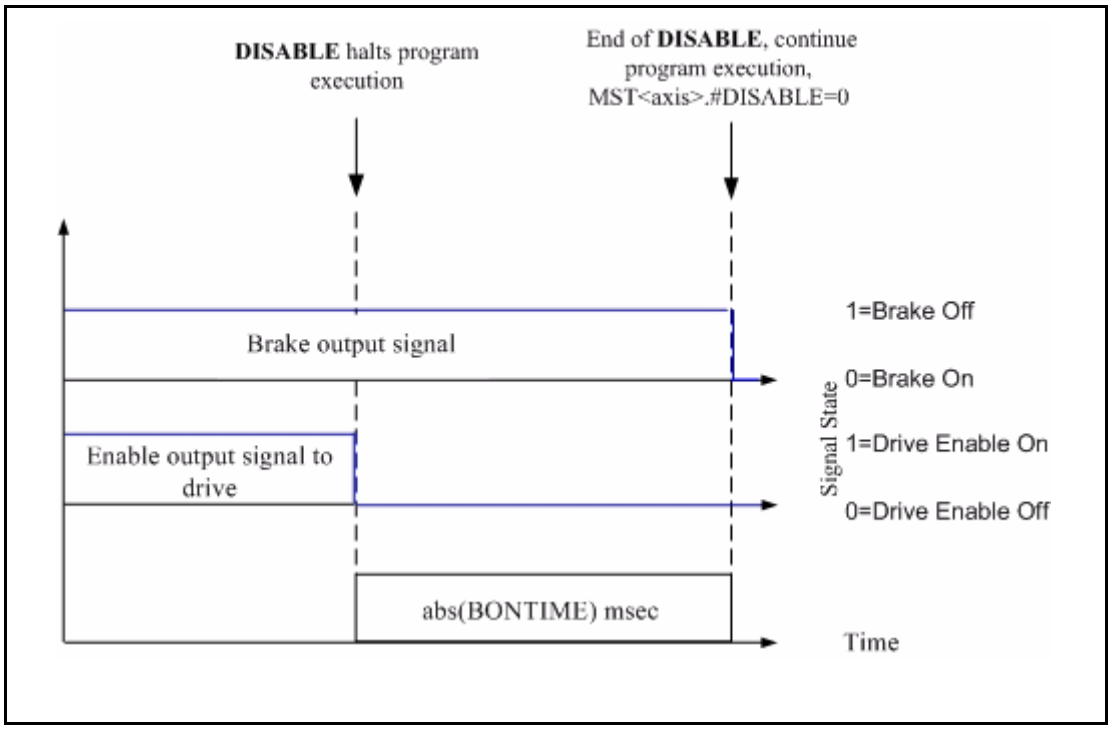


Figure 3 DISABLE and Mechanical Brake Output Process - Negative BONTIME

2.1.6 ENABLE/ENABLEALL

Description

ENABLE activates one or more drives. After **ENABLE**, the motor starts following the reference position (**RPOS**) and axis faults are enabled. **ENABLEALL** activates all drives.

Syntax

ENABLE|**ENABLEALL** *axis_list*

Arguments

axis_list	Axis or list of axes, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.

Comments

Motor specification is a single axis like 0 or 13, a string of axes enclosed in parentheses and separated by commas, for example: (0,1,13), or the keyword: **ALL** for all axes.

Related ACSPL+ Commands

DISABLE

Related ACSPL+ Variables

ENTIME, **MFLAGS**<*axis*>.#ENMOD

COM Library Methods

Enable, EnableM

C Library Functions

acsc_Enable, acsc_EnableM

Examples

Figure 4 and **Figure 5** illustrate the **ENABLE** process using positive and negative **BOFFTIME** values.

Example 1:

```
ENABLE 0                !Enable 0 axis.
```

Example 2:

```
ENABLE (0,1)            !Enable 0 and 1 axes.
```

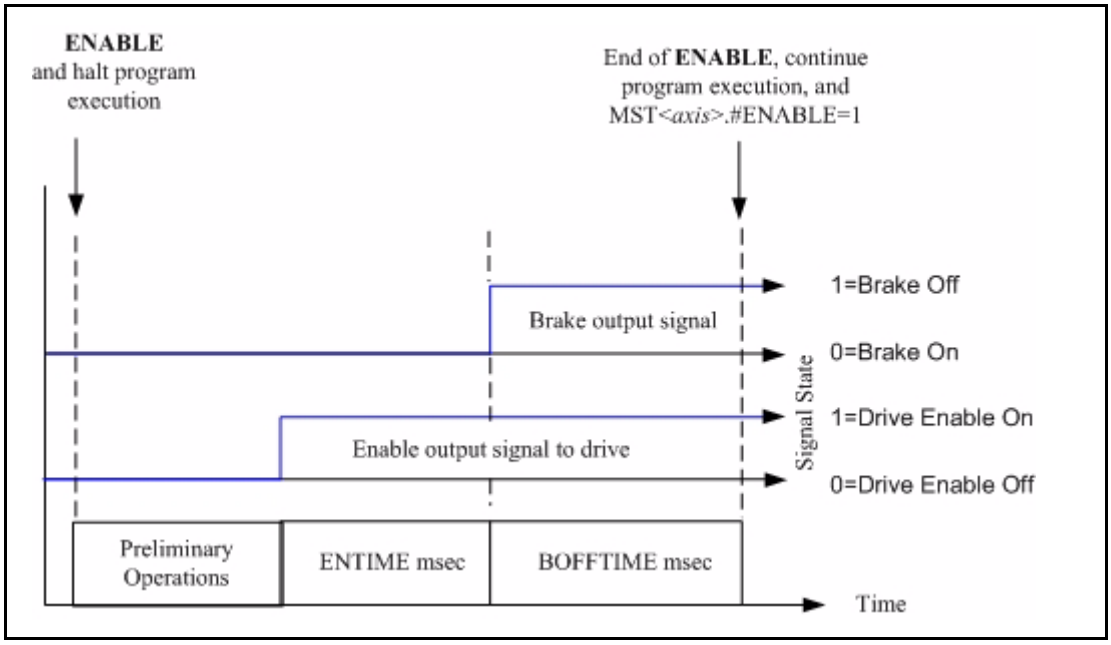


Figure 4 ENABLE and Mechanical Brake Output Process - Positive BOFFTIME

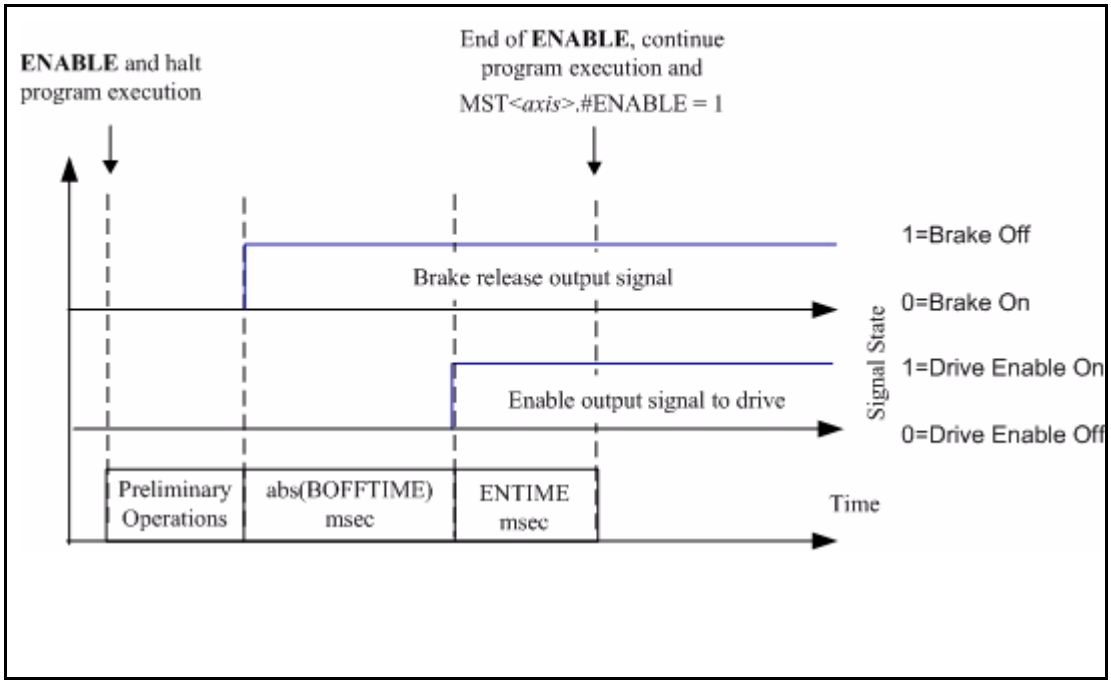


Figure 5 ENABLE and Mechanical Brake Output Process - Negative BOFFTIME

2.1.7 FCLEAR

Description

FCLEAR clears the **FAULT** variable and the results of the previous fault stored in **MERR**.

Syntax

FCLEAR *axis_list*

Arguments

axis_list	Axis or list of axes, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.

Comments

1. Motor specification is a single axis like 0 or 13, a string of axes enclosed in parentheses and separated by commas, for example: (0,1,13), or the keyword: ALL for all axes.
2. If the axis designation is omitted the command clears the system faults. If an axis is specified, the command clears the **FAULT** and **MERR** components for the specified axes. However, if the reason for the fault is still active, the controller will immediately set the fault again following **FCLEAR**.
3. If one of the cleared faults is an encoder error, **FCLEAR** also resets the feedback position to zero.

Related ACSPL+ Variables

MERR, **FAULT**

COM Library Methods

FaultClear, FalutClearM

C Library Functions

acsc_FaultClear, acsc_FaultClearM

Examples

Example 1:

```
FCLEAR (0,1)          !Clear FAULT and MERR variables for 0 and 1 axes
```

Example 2:

```
FCLEAR ALL            !Clear FAULT and MERR variables for all axes
```

2.1.8 GO

Description
GO starts a motion that has been created using the /w (wait) suffix.

Syntax
GO *axis_list*

Arguments

axis_list	Axis or list of axes, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.

Comments
1. Motor specification is a single axis like 0 or 13, a string of axes enclosed in parentheses and separated by commas, for example: (0,1,13), or the keyword: ALL for all axes.
2. Where **GO** specifies a single axis, that axis may not be included in any group. **GO** starts the last created motion for the same axis. If the motion was not created, or has been started before, the command has no effect.
3. Where **GO** specifies a leading axis in a group, **GO** starts the last created motion for the same axis group. If the motion was not created, or has been started before, the command has no effect.

COM Library Methods
Go, GoM

C Library Functions
acsc_Go, acsc_GoM

Related ACSPL+ Commands
HALT, **MSEG...ENDS**, **JOG**, **MPTP...ENDS**, **PATH...ENDS**, **PTP**, **PVSPLINE...ENDS**, **SLAVE**, **TRACK**

Example

```
PTP/w (0,1), 1000, 1000 !Create PTP motion, but do not start
PTP/w 3, 8000           !Create PTP motion, but do not start
GO (0,1)                !Start both motions at the same time
```

2.1.9 GROUP

Description
GROUP defines an axis group for coordinate multi-axis motion. The first axis in the axes list is the leading axis. The motion parameters of the leading axis become the default motion parameters for all axes in the group. Motion on all axes in a group will start and conclude at the same time.

Syntax
GROUP *axis_list*

Arguments

axis_list	Axis or list of axes, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.

Comments
An axis can belong to only one group at a time. If the application requires restructuring the axes, it must split the existing group and only then create the new group.

Related ACSPL+ Variables
VEL, **ACC**, **DEC**, **JERK**, **KDEC**, **GVEC**, **GVEL**, **GACC**, **GJERK**

Related ACSPL+ Commands
SPLIT

COM Library Command
Group

C Library Command
acsc_group

Example

```
GROUP (0,1)              !Creates an axis group that includes axes 0 and 1.
PTP (0,1), 1000, 10000   !PTP axis 0 to 1000, and axis 1 to 10000.
```

2.1.10 HALT

Description

In single axis motion, **HALT** terminates currently executed motion and clears all other motions waiting in the axis motion queue. The deceleration profile is defined by **DEC** (deceleration variable).

In group motion, **HALT** terminates currently executed motion of all group axes, and clears all other motions waiting in the axes motion queues.The deceleration profile is defined by the **DEC** (deceleration) variable of the leading axis.

Syntax

HALT *axis_list[,reason]*

Arguments

axis_list	List of axes to be halted.
reason	An optional argument, which must be an integer constant or expression that equates to an integer, and specifies a cause why the axis was halted. If the parameter is specified, its value is stored in the **AERR** variable. If the parameter is omitted, **AERR** stores 5003 after **HALT**.

Comments

HALT ALL terminates the motion of all axes.

Related ACSPL+ Commands

KILL

COM Library Methods

Halt, HaltM

C Library Functions

acsc_Halt, acsc_HaltM

Examples

Example 1:

```
HALT 0                  !Terminates 0 axis motion, the deceleration is
                        !according to DEC(0)
```

Example 2:

```
HALT (0,2)              !Terminates currently executed motion on axes
                        !0 and 2.
```

Example 3:

```
HALT ALL                !Terminates currently executed motion on all axes.
```

2.1.11 IMM

Description

In single axis motion, **IMM** provides on-the-fly changes of the following motion parameters:

- **VEL** (Velocity)
- **ACC** (Acceleration)
- **DEC** (Deceleration)
- **JERK** (Jerk)

IMM affects the motion in progress and all motions waiting in the corresponding motion queue.

In group motion, **IMM** provides on-the-fly changes for the motion parameters of the leading axis only.

Syntax

IMM *axis_motion parameter=value or formula*

Arguments

axis_motion parameter	The motion parameter with the specified axis (valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus)1.
value or formula	User specified value or formula.

Related ACSPL+ Variables

VEL, **ACC**, **DEC**, **JERK**

COM Library Methods

SetAccelerationImm, SetDecelerationImm, SetJerkImm, SetKillDecelerationImm, SetVelocityImm

C Library Functions

acsc_SetAccelerationImm, acsc_SetDecelerationImm, acsc_SetJerkImm, acsc_SetKillDecelerationImm, acsc_SetVelocityImm

Example

```
IMM VEL(0)=5000         !Immediately change the 0 axis velocity to 5000
```

2.1.12 KILL

Description

Use **KILL** after a safety event to decelerate and stop the axis faster than during normal deceleration and stop.

In single axis motion, **KILL** terminates currently executed motion and clears all other motions waiting in the axis motion queue. The deceleration profile uses a second-order deceleration profile and the **KDEC** (kill deceleration) value.

In group motion, **KILL** terminates currently executed motion only for the specified axes, and clears all other motions waiting in the axis/axes motion queue.The deceleration profile uses a second-order deceleration profile and the **KDEC** (kill deceleration) variable of each axis.

Syntax

KILL *axis_list[,reason]*

Arguments

axis_list	List of axes to be killed, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
reason	An optional argument, which must be an integer constant or expression that equates to an integer, and specifies a cause why the axis was killed. If the parameter is specified, its value is stored in the **MERR** variable. If the parameter is omitted, **MERR** stores zero after **KILL**.

Comments

1. **KILL ALL** terminates the motion of all axes. The deceleration profile is defined by the **KDEC** (kill deceleration) variable of each axis.
2. If several sequential **KILL** operations specify different causes for the same motor, only the first *cause* is stored in **MERR** and all subsequent causes are ignored.
3. A *cause* stored in **MERR** is cleared by **FCLEAR** or **ENABLE/ENABLEALL**.

Related ACSPL+ Commands

HALT, **FCLEAR**, **ENABLE/ENABLEALL**

Related ACSPL+ Variables

MERR, **KDEC**

COM Library Methods

Kill, KillAll

C Library Functions

acsc_Kill, acsc_KillAll

Examples

Example 1:

```
KILL 1                  !Kill axis 1 deceleration is according to KDEC(1)
```

Example 2:

```
KILL 0, 5011            !Kill 0 axis, store 5011 as the reason.
                        !Code 5011 corresponds to left limit error;
                        !therefore the 0 motor will be reported as
                        !disabled due to fault involving left limit.
```

Example 3

```
KILL (0,1,2)            !Kill 0, 1 and 2 axes according to the KDEC
                        !of each specified axis.
```

Example 4

```
KILL ALL, 6100          !Kills all axes, and stores the cause in MERR.
                        !(6100 is the code for a user-defined cause.)
```

2.1.13 SAFETYCONF

Description

SAFETYCONF configures fault processing for one or more axes, by disabling the default response to a defined axis **FAULT**, and performs one of the following responses:

- Ignore the interrupt
- Kill the motion
- Disable the axis
- Kill the motion and then disable the axis

Syntax

SAFETYCONF *axis_list, fault_name, "conf_string"*

Arguments

axis_list	Axis or list of axes, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
fault_name	Any axis fault name like #LL for left limit.
conf_string	A string enclosed in double quotation marks with one or more of the following characters determines the action of **SAFETYCONF**: ❑ K (**KILL**) ❑ D (**DISABLE**) ❑ KD (**KILL-DISABLE**) ❑ + when a fault occurs in any member of the axis_list, the fault response applies to all axes of the controller (each axis fault response can be unique) ❑ - applies **FMASK**<*axis*>.***#fault-name*** = 0 for all axes of the controller

Comments

If an empty string is specified, fault detection is enabled, but the controller has no response to the fault. However, an autoroutine can intercept the fault and provide a response.

Related ACSPL+ Commands

The **#SC** immediate command shows the current fault response configuration for all axes.

Examples

Example 1:

```
SAFETYCONF 0,#PE,"K"     !The 0 motion will be killed if the 0
                         !Position Error fault occurs.
```

Example 2:

```
SAFETYCONF               !Changes the response to the Left Limit fault of
(3,5),#LL,"KD"
                         !3 and 5 axes to KILL and then DISABLE.
```

Example 3:

```
SAFETYCONF ALL,#DRIVE,"D+"   !All axes will be disabled if the Drive
                             !Alarm fault occurs in any axis.
```

Example 4:

```
SAFETYCONF ALL,#VL,"-"   !Velocity Limit fault will be masked for
                         !all axes.
```

2.1.14 SAFETYGROUP

Description
SAFETYGROUP activates the fault response for all axes in the axis_list when any axis triggers the fault, and manages the axes as a block in response to **KILL** and **DISABLE**.

To cancel the defined **SAFETYGROUP**, send the command again with only the first axis as the axis_list.

Syntax
SAFETYGROUP *axis_list*

Arguments

axis_list	List of axes, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.

COM Library Methods
GetSafetyInputPort

C Library Functions
acsc_GetSafetyInputPort

Examples
Example 1

```
SAFETYGROUP (0,1,5)      !Creates safety group for axes 0, 1 and 5.
```

Example 2

```
SAFETYGROUP 0            !Cancels the previously created safety group for
                         !axes 0, 1 and 5.
```

2.1.15 SET

Description
SET defines the current value of either feedback (**FPOS**), reference (**RPOS**), or axis (**APOS**) position. **SET** can be initiated when the axis is disabled, or on-the-fly. **APOS** and **FPOS** are updated automatically when **SET** is specified for **RPOS**,

If a non-default **CONNECT** is used, assign different values to **APOS** and **RPOS**.

Related ACSPL+ Variables
RPOS

Syntax
SET axis_RPOS=*value or formula*

Arguments

axis_RPOS	The reference position of the specified axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
=	The assignment operator
value or formula	User-defined value or formula

COM Library Methods
SetRPosition, GetRPosition

C Library Functions
acsc_SetRPosition, acsc_GetRPosition

Example

```
SET RPOS(0)=300         !Axis 0 RPOS = 300
```

2.1.16 SPLIT

Description
SPLIT breaks down a group created using **GROUP** by designating any axis in the axis list.
SPLIT ALL breaks down all groups.

Note	
	If the **SPLIT** *command specifying an axis that is currently in motion is executed within the buffer, the buffer execution is suspended until the motion is completed. However, if the* **SPLIT** *command is sent from the host or as an Immediate command, it returns error 3087: "Command cannot be executed while the axis is in motion".*

Syntax
SPLIT *axis_list*

Arguments

axis_list	Axis or list of axes, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.

Related ACSPL+ Commands
GROUP

COM Library Methods
Split, SplitAll

C Library Functions
acsc_Split, acsc_SplitAll

Examples

Example 1

```
GROUP (0,1,5)          !Create an axis group including axes 0, 1 and 5.
GROUP (2,3,7)          !Create an axis group including axes 2, 3 and 7.
SPLIT 0                !Breaks down axis group 0, 1 and 5.
```

Example 2

```
GROUP (0,1,5)          !Create an axis group including axes 0, 1 and 5.
GROUP (2,3,7)          !Create an axis group including axes 2, 3 and 7.
SPLIT ALL              !Breaks down axis group 0, 1 and 5
                       !and group 2, 3 and 7.
```

2.2 Interactive Commands

The Interactive commands are:

Command	Action
DISP	Builds a string and sends it to the default communication channel.
INTERRUPT	Causes an interrupt that can be intercepted by the host.
SEND	Same as **DISP**, but also specifies the communication channel or channels.
TRIGGER	Specifies a triggering condition. Once the condition is satisfied, the controller issues an interrupt to the host computer.

2.2.1 DISP

Description

DISP builds an ASCII output string and sends it to a communication channel. The ASCII output can include text segments and variable values defined in various format displays. The output string is sent to the default communication channel defined by the standard variable **DISPCH**.

Syntax

DISP *argument [, argument. . .]*

Arguments

argument	An expression or string where: **Expression:** ACSPL+ expression as one or more variables **String:** "[text] [escape-sequence] [format-specifier]" String must be enclosed with double quotation marks.
[, argument. . .]	Optional subsequent arguments.

Command Options

An input string can include one or more of the following:

- Text

  Escape Sequence - appears in the output string as a non-printing character or other specified character, as described in **Table 4**.

- **Formatting Specification** - determines how the results of an expression that follows the input string is formatted in the output string.

Table 4 DISP Command Option Escape Sequences

Escape Sequence	Added Character to Output String
\r	Carriage return - 0x0d
\s	Avoid carriage return
\n	New line - 0x0a
\t	Horizontal tab - 0x09
\xHH	Any ASCII character where **HH** represents the ASCII code of the character.

Note	
	The output format specification syntax adheres to a restricted version of the C language syntax.

The format specification syntax is:

```
% [width] [.precision] type
```

where:

[width]	Optional specification for the minimum number of characters in the output. If ***width*** is smaller than the number of digits of a displayed number, the specified ***width*** is ignored, and the displayed number includes all digits.
[.precision]	Optional number that specifies the maximum number of characters printed for all or part of the output field, or the minimum number of digits printed for integer values.
type	Required character that determines whether the associated argument is interpreted as a character, a string, or a number, described in **Table 5**.

Table 5 Type Characters

Type	Output Format
d	Signed decimal integer
u	Unsigned decimal integer
x	Unsigned hexadecimal integer, using "abcdef"
X	Unsigned hexadecimal integer, using "ABCDEF"
e	Signed value having the form [-]d.dddd e [sign]ddd where d is a single decimal digit, dddd is one or more decimal digits, ddd is exactly three decimal digits, and sign is + or -.
E	Identical to the **e** format except that **E** rather than **e** introduces the exponent.
f	Signed value having the form [-]dddd.dddd, where dddd is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision.
g	Signed value printed in **f** or **e** format, whichever is more compact for the given value and precision. The e format is used only when the exponent of the value is less than -4 or greater than or equal to the precision argument. Trailing zeros are truncated, and the decimal point appears only if one or more digits follow it.
G	Identical to the **g** format, except that **E**, rather than **e**, introduces the exponent (where appropriate).

Comments

1. If an input string argument contains **n** format specifiers, the specifiers apply to the **n** subsequent expression arguments.
2. **DISP** processes arguments from left to right, as follows:
   a) **Expressions**: The expression is evaluated and the ASCII representation of the result is placed in the output string. The format of the result is determined by the formatting specifications (if any) in the input string.
   b) **Input strings**: Text is sent as-is to the output string. Escape sequences are replaced by the ASCII codes that they represent. Formatting specifications are applied to the results of any expressions that follow the string.
3. **DISP** cannot be used from the SPiiPlus MMI Application Studio **Communication Terminal,** only from a program buffer.
4. **DISP** can only display the value of a single element of an array.

Related ACSPL+ Commands

SEND

Related ACSPL+ Variables
DISPCH

COM Library Methods
OpenMessageBuffer, GetSingleMessage

C Library Functions
acsc_OpenMessageBuffer, acsc_GetMessage

Examples
Example 1:

```
DISP "%15.10f",FPOS0     !Display FPOS0 in 15 digits with 10 digits
                         !following the decimal point.
                         !Output: 997.2936183303
STOP
```

Example 2:

```
DISP "0 FVEL=%15.10f", FVEL0
                         !Output: 0 FVEL= 997.2936183303
STOP
```

Example 3:

```
DISP "%1i",IN0.2         !Output: the current state of IN0.2 as
                         !one digit 0 or 1.
STOP
```

Example 4:

```
DISP "IN0 as hex: %04X",IN0
                         !Output: IN0 as hex: 0A1D
STOP
```

Example 5:

```
DISP "IN0.0-3 as binary: %1u%1u%1u%1u", IN0.0,IN0.1,IN0.2,IN0.3
                         !Output: IN0.0-3 as binary: 0110
STOP
```

Example 6:

```
DISP "Elapsed time is:  ", TIME
                        !Output: Elapsed time is:  4.93258E+006
STOP
```

Example 7:

```
DISP "Expression, default formating:", FPOS0*2+FPOS1+5 , FPOS1
                        !Output: Expression, default formatting: 6.28657
                        !0.286485
STOP
```

Example 8:

```
REAL AXIS_NAME ; AXIS_NAME=0 ;
DISP "Axis",AXIS_NAME," was disabled due to error code", MERR(AXIS_NAME)
                        !Display the reason of axis disable due to a fault.
                        !Output: Axis 0  was disabled due to error code 0
STOP
```

Example 9:

```
DISP "%5i\r", FPOS0, "%5i", FPOS1
                        !Standard format, minimum 5 positions, no decimals,
                        !and a carriage return between the two values.
                        !Output:
                        !711
                        !2024
STOP
```

Example 10:

```
DISP "Hexadecimal format: %08X", MFLAGS(0), "  and also %08x", MFLAGS(0)
                        !Hexadecimal format, minimum 8 positions, capital
                        !letters or lower case letters.
                        !Output: Hexadecimal format: 002A2300 and also
                        !002a2300
STOP
```

Example 11:

```
DISP "0 FPOS: %15.3e", FPOS0," and 1 FPOS:  %15.3e", FPOS1
                        !Scientific format, minimum 15 positions, 3 decimals
                        !of FPOS0 and FPOS1.
                        !Output:
                        !0 FPOS: 5.000e-001 and
                        !1 FPOS: 2.865e-001
STOP
```

Example 12:

```
REAL AAA;               !Standard or scientific format, small letters or
                        !capital letters.
AAA=10;                 !Assigning integer value to AAA
DISP "%g", AAA;         !Output: 10
AAA=1e9                 !Assigning Hex value to AAA
DISP "%g", AAA          !Output: 1e+009
DISP "%G", AAA          !Output: 1E+009
STOP
```

2.2.2 INTERRUPT

Description
INTERRUPT causes an unconditional trigger that is intercepted by the host. Once a program executes **INTERRUPT**, the interrupt signal is sent to the host application. This interrupt is detected by the COM library **EnableEvent** or C Library **acsc_SetCallBack** functions which then call interrupt type **ACSC_INTR_ACSPL_PROGRAM**.

Syntax
INTERRUPT

Related ACSPL+ Commands
TRIGGER

COM Library Methods
EnableEvent, DisableEvent, SetCallbackMask, GetCallbackMask

C Library Functions
acsc_InstallCallback, acsc_SetCallbackMask, acsc_GetCallbackMask

Examples
Example 1:

INTERRUPT used in an ACSPL+ program:

```
ENABLE 0                !Enable axis 0
SET FPOS0=0             !Set axis 0 feedback position = 0
PTP 0, 1000             !ACSPL+ executes a PTP motion.
TILL MST(0).#MOVE=0     !The motor reaches the destination point
                        !and stops.
INTERRUPT               !INTERRUPT is sent to the host application.
STOP
```

Example 2:

INTERRUPT used in a Host application:

```
SET ch = New Channel    !Initialize ch as a COM library object
CALL ch.EnableEvent(ch.ACSC_INTR_ACSPLPROGRAM)
                        !Enable INTERRUPT as an event. The host application
                        !waits for an interrupt from the controller initiated
                        !by INTERRUPT from within the ACSPL+ program.
Private Sub ch_ACSPLPROGRAM(ByVal Param As Long) MsgBox ("Motion is Stopped")
                        !When an interrupt occurs, launch a message box
                        !displaying “Motion is Stopped”
EndSub
```

2.2.3 SEND

Description

SEND is the same as **DISP**, but also specifies the communication channel for the output string.

Syntax

SEND *channel_number, argument [, argument. . .]*

Arguments

channel-number	An integer that defines the communication channel to which the message will be sent, as follows: -2: All channels -1: Last used channel 1: Serial Port 1 2: Serial Port 2 6: Ethernet network (TCP) 7: Ethernet network (TCP) 8: Ethernet network (TCP) 9: Ethernet network (TCP) 10: Ethernet point-to-point (UDP) 16: MODBUS Slave 36: Ethernet network (TCP) 37: Ethernet network (TCP) 38: Ethernet network (TCP) 39: Ethernet network (TCP)
argument	An expression or string where: **Expression:** ACSPL+ expression as one or more variables **String:** "[text] [escape-sequence] [format-specifier]" String must be enclosed with double quotation marks.
[, argument. . .]	Optional subsequent arguments.

Command Options

For a list of **Command Options**, relevant **Comments**, and **Examples**, see **DISP**.

Related ACSPL+ Commands

DISP

Related ACSPL+ Variables

DISPCH

COM Library Methods

Send

C Library Functions

acsc_Send

2.2.4 TRIGGER

Description

TRIGGER specifies a triggering condition. Once the condition is satisfied, the controller issues an interrupt to the host computer, as follows:

1. Sets **AST**<*axis*>**.#TRIGGER** = 0
2. Examines the triggering condition every MPU cycle

Once the condition is satisfied, the controller performs the following:

1. Sets **AST**<*axis*>**.#TRIGGER** = 1
2. Produces an interrupt to the host application (software interrupt 10, enabled by **IENA**.26).

The controller continues calculating the **TRIGGER** expression until another **TRIGGER** command is executed in the same channel. Each time the expression changes its value from zero to non-zero, the controller sets **AST**<*axis*>**.#TRIGGER** = 1 and causes an interrupt.

Note	*Full application of the* **TRIGGER** *command to channels greater than 7 is not currently supported.*

Syntax

TRIGGER *channel, expression[, timeout]*

Arguments

channel	An integer number from 0 to 7 that specifies the trigger's sequential number. The number defines the **AST** element where the triggering bit will be set and defines the bit in the interrupt tag that is automatically sent to the host application as an interrupt.
expression	Specifies the triggering condition. After **TRIGGER** is executed, the controller checks the expression condition each MPU cycle. Triggering occurs when the expression condition is satisfied. If the argument is omitted, triggering in the specified channel is disabled.
timeout	Specifies triggering timeout in milliseconds. A positive number specifies the time allowed for the triggering condition to be satisfied. If the timeout has elapsed and the triggering condition has not been satisfied, the controller unconditionally raises the trigger bit. After any triggering, the controller resets timeout counting to zero. If the argument is omitted, triggering works without timeout.

Table 6 Channel Designation for TRIGGER

Channel	Triggering Bit	Hexadecimal Interrupt Tag (Software Interrupt 10)
0	AST0.11	0x00000001
1	AST1.11	0x00000002
2	AST2.11	0x00000004
3	AST3.11	0x00000008
4	AST4.11	0x00000010
5	AST5.11	0x00000020
6	AST6.11	0x00000040
7	AST7.11	0x00000080

Related ACSPL+ Commands
INTERRUPT

Related ACSPL+ Variables
IENA, **AST**

COM Library Methods
GetCallbackMask

C Library Functions
acsc_GetCallbackMask

Example

```
TRIGGER 1, MST(0).#MOVE=0, 3000
                        !1 – once the triggering condition is satisfied,
                        !the triggering bit AST1.#TRIGGER will be set
                        !to "1", and the interrupt tag to the host
                        !application is 0x00000002.
                        !MST(0).#MOVE=0 – the triggering condition.
                        !Actuate trigger interrupt when the motor in
                        !the X axis is in position (not moving).
                        !3000 – check the triggering condition for
                        !3000 msec. If the triggering condition is not
                        !satisfied after 3000 msec, then set the
                        !triggering bit AST(1).TRIGGER to "1".
```

2.3 Miscellaneous Commands

The Miscellaneous commands are:

Command	Action
ASSIGNFINS	Assigns PEG inputs
ASSIGNPEG	Assigns PEG engines to encoders.
ASSIGNPOUTS	Assigns PEG outputs
AXISDEF	Establishes an axis alias.
DC	Activates data collection.
PEG_I	Activates incremental PEG, where pulses are generated at predefined, fixed position intervals.
PEG_R	Activates random, table-based PEG where predefined pulses are generated at pre-defined positions
READ	Reads an array from a file in the flash memory.
SETPROTECTION	Changes protection attribute for a variable.
SPDC	Activates data collection from a SP variable.

Command	Action
STARTPEG	Starts PEG motion for specified axis.
STOPDC	Terminates data collection.
STOPPEG	Terminates PEG motion.
WRITE	Writes an array to a file in the flash memory.

2.3.1 AXISDEF

Description

The **AXISDEF** command enables the user to assign an alias to one or more axes. Once assigned, the user can use the alias throughout the program in any command requiring an axis argument.

Syntax

AXISDEF *axis_alias = axis*

Arguments

axis_alias	Any string with the following restrictions: ❑ Only one name can be defined for one axis, that is, different names cannot be used for the same axis. ❑ The names must be unique, i.e., two axes cannot be defined with the same name. ❑ An axis name must not be the same of any other variable name, label, keyword, etc. A compilation error occurs if one of the above restrictions is not satisfied.
axis	The axis number, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.

Comments

The **AXISDEF** command can be repeated many times to define all required aliases.

The axis name must be defined either in the D-Buffer, or in one of the buffers in which it is used. In any case, the axis definition has global scope (the definitions of the same axis in a different program must be identical as applies to all global variables).

The controller supports prefix indexing for user-defined and legacy axes. However, for new applications it is recommended using explicit indexing and providing names as symbolic constants.

Related ACSPL+ Commands

None

COM Library Methods

None

C Library Functions

None

Examples

1. An axis name can be used in expressions as a symbolic constant. For example, given the program includes the declaration:

   ```
   AXISDEF Q=3
   ```

   the following command

   ```
   VEL(Q)=1000;
   ```

   assigns 1000 to the required velocity of axis 3.

2. As user variables, axis-related standard variables accept explicit indexing. However, axis-related standard variables also accept postfix indexing. For example, given a program that includes the declaration:

   ```
   AXISDEF Q=3, X1=1, X2= 2
   ```

   the following table provides examples of index formats:

Explicit Indexing	Postfix Indexing
VEL(3) or VEL(Q)	VEL3
ACC(1) or ACC(X1)	ACC1
SLVKI(2) or SLVKI(X2)	SLVKI2

2.3.2 ASSIGNPEG

Description

Position Event Generation (**PEG**) is a fast output based on an axis position with sub microsecond delay. The **ASSIGNPEG** function is used for engine-to-encoder assignment as well as for the additional digital outputs assignment for use as PEG state and PEG pulse outputs.

Syntax

ASSIGNPEG *axis, engines_to_encoders_code, dig_out_assign_code*

Arguments

axis	The axis index, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
engines_to_encoders_code	Bit code for engines-to-encoders mapping according to **Table 7** and **Table 8**
dig_out_assign_code	Digital outputs assignment to use as PEG state and PEG pulse outputs according to **Table 9** and **Table 10**.

Model	
	There can be either 4 or 8 Encoder connectors, depending on the controller model. ❑ *Encoders:* **0(X)**, **1(Y)**, **2(A)**, *and* **3(B)** *are controlled by Servo Processor 0* ❑ *Encoders:* **4(Z)**, **5(T)**, **6(C)**, *and* **7(D)** *are controlled by Servo Processor 1*

Table 7 Mapping PEG Engines to Encoders (Servo Processor 0)

Bit Code	**Servo Processor 0**			
	Encoder 0(X)	Encoder 1(Y)	Encoder 2(A)	Encoder 3(B)
000 (default)	PEG0	PEG1	PEG2	no
001	PEG0	PEG1	no	PEG2
010	PEG0 PEG2	PEG1	no	no
011	PEG0	PEG1 PEG2	no	no
100	PEG0 PEG1 PEG2	no	no	no
101	no	PEG0 PEG1 PEG2	no	no

Table 8 Mapping PEG Engines to Encoders (Servo Processor 1) (page 1 of 2)

Bit Code	**Servo Processor 1**			
	Encoder 4(Z)	Encoder 5(T)	Encoder 6(C)	Encoder 7(D)
000 (default)	PEG4	PEG5	PEG6	no
001	PEG4	PEG5	no	PEG6
010	PEG4 PEG6	PEG5	no	no
011	PEG4	PEG5 PEG6	no	no
100	PEG4 PEG5 PEG6	no	no	no

Table 8 Mapping PEG Engines to Encoders (Servo Processor 1) (page 2 of 2)

Bit Code	Servo Processor 1			
101	no	PEG4 PEG5 PEG6	no	no

Note

*In **Table 7** and **Table 8** the* **Bit Code** *affects all of the connectors in the row, for example, the* **Bit Code***: 001 for those connectors controlled by Servo Processor 0 performs the following assignments:*

PEG0 → Encoder 0

PEG1 → Encoder 1

PEG2 → Encoder 3

Table 9 Digital Outputs Assignment for Use as PEG State and PEG Pulse Outputs (Servo Processor 0)

Bit Code	Servo Processor 0			
	GP out 0	GP out 1	GP out 2	GP out 3
0000 (default)	GP out 0	GP out 1	GP out 2	GP out 3
0001	PEG_Pulse(0)	GP out 1	GP out 2	GP out 3
0010	GP out 0	PEG_Pulse(2)	GP out 2	GP out 3
0011	GP out 0	GP out 1	PEG_Pulse(1)	GP out 3
0100	GP out 0	GP out 1	GP out 2	Reserved
0101	GP out 0	PEG_Pulse(2)	GP out 2	Reserved
0110	PEG_Pulse(0)	GP out 1	PEG_Pulse(1)	GP out 3
0111	PEG_Pulse(0)	PEG_Pulse(2)	PEG_Pulse(1)	Reserved
1000	Reserved	Reserved	Reserved	Reserved
1001	Reserved	Reserved	Reserved	Reserved
1010	Reserved	Reserved	Reserved	Reserved
1011	Reserved	Reserved	Reserved	Reserved
1100	Reserved	Reserved	Reserved	Reserved
1101	Reserved	Reserved	Reserved	Reserved
1110	Reserved	Reserved	Reserved	Reserved
1111	Reserved	Reserved	Reserved	Reserved

Table 10 Digital Outputs Assignment for Use as PEG State and PEG Pulse Outputs (Servo Processor 1)

Bit Code	Servo Processor 1			
	GP out 4	GP out 5	GP out 6	GP out 7
0000 (default)	GP out 4	GP out 5	GP out 6	GP out 7
0001	PEG_Pulse(4)	GP out 5	GP out 6	GP out 7
0010	GP out 4	PEG_Pulse(6)	GP out 6	GP out 7
0011	GP out 4	GP out 5	PEG_Pulse(5)	GP out 7
0100	GP out 4	GP out 5	GP out 6	Reserved
0101	GP out 4	PEG_Pulse(6)	GP out 6	Reserved
0110	PEG_Pulse(4)	GP out 5	PEG_Pulse(5)	GP out 7
0111	PEG_Pulse(4)	PEG_Pulse(6)	PEG_Pulse(5)	Reserved
1000	Reserved	Reserved	Reserved	Reserved
1001	Reserved	Reserved	Reserved	Reserved
1010	Reserved	Reserved	Reserved	Reserved
1011	Reserved	Reserved	Reserved	Reserved
1100	Reserved	Reserved	Reserved	Reserved
1101	Reserved	Reserved	Reserved	Reserved
1110	Reserved	Reserved	Reserved	Reserved
1111	Reserved	Reserved	Reserved	Reserved

Note

*In **Table 9** and **Table 10** the* **Bit Code** *affects the entire row, for example, for those connectors controlled by Servo Processor 0,* **0110** *switches* **PEG_Pulse(0)** *to* **GP out 0** *and* **PEG_Pulse(2)** *to* **GP out 1**. *The same* **Bit Code** *applied to those connectors controlled by Servo Processor 1 switches* **PEG_Pulse(4)** *to* **GP out 4** *and* **PEG_Pulse(5)** *to* **GP out 6**.

Related ACSPL+ Commands
ASSIGNPOUTS, **PEG_I**, **PEG_R**, **STOPPEG**

COM Library Methods
None

C Library Functions
None

2.3.3 ASSIGNPOUTS

Description

The **ASSIGNPOUTS** function is used for setting input pins assignment and mapping between **FGP_OUT** signals to the bits of the **OUT**variable.

Each PEG engine has 1 PEG pulse output and 4 state outputs for a total of 5 outputs per PEG engine and a total of 30 outputs for the whole PEG generator. The controller supports 10 physical output pins that can be assigned to the PEG generator. The user defines which 10 outputs (of the 30) of the PEG generator are assigned to the 10 available physical output pins. Some of the output pins are shared between the PEG and the HSSI.

The tables below define how each of the 30 outputs of the 6 PEG engines can be routed to the 10 physical output pins - 4 PEG out signals, 3 PEG state signals, and 3 HSSI signals. It needs to be noted that some of the signals cannot be routed to the physical pins.

Syntax

ASSIGNPOUTS *axis, output_index, bit_code*

Arguments

axis	The axis index, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
output_index	Output Index (0 for **OUT_0**, 1 for **OUT_1**, ..., 9 for **OUT_9**)
bit_code	Bit code for engines outputs to physical outputs mapping according to **Table 11** and **Table 12**.

Note	
	Table 11 *provides the mapping of the PEG engines outputs (30 signals) to the physical outputs of the first group of 5 out of the 10 output pin (Servo Processor 0)s, and* ***Table 12*** *provides to mapping to the second group of 5 out of 10 output pins (Servo Processor 1).*

Table 11 Mapping of Engine Outputs to Physical Outputs (Servo Processor 0)

Bit Code	Out_4 (HSSI1_DO)	Out_3 (T_PEG)	Out_2 (Z_PEG)	Out_1 (Y_PEG)	Out_0 (X_PEG)
000 (default)	HSSI1_DO	PEG5_pulse	PEG4_Pulse	PEG1_pulse	PEG0_Pulse
001	PEG0_OUT1	PEG0_OUT0	PEG2_Pulse	PEG1_OUT0	PEG4_OUT0
010	PEG2_OUT1	PEG2_OUT0	PEG1_OUT1	Reserved	Reserved
011	Reserved	Reserved	Reserved	Reserved	Reserved
100	Reserved	Reserved	Reserved	Reserved	Reserved
111	Reserved	FGP_OUT3	FGP_OUT2	FGP_OUT1	FGP_OUT0

Table 12 Mapping of Engine Outputs to Physical Outputs (Servo Processor 1)

Bit Code	Out_9 HSSI0_CON	Out_8 HSSI0_DO	Out_7 X_STATE2	Out_6 X_STATE1	Out_5 X_STATE0
000 (default)	HSSI0_CON	HSSI0_DO	PEG0_OUT2	PEG0_OUT1	PEG0_OUT0
001	PEG0_OUT3	PEG0_OUT2	PEG2_OUT0	PEG1_OUT1	PEG1_OUT0
010	PEG2_OUT1	PEG6_OUT0	PEG6_Pulse	PEG5_Pulse	PEG4_Pulse
011	PEG6_OUT1	Reserved	PEG5_OUT1	PEG5_OUT0	Reserved
100	Reserved	Reserved	Reserved	Reserved	Reserved
111	Reserved	Reserved	FGP_OUT6	FGP_OUT5	FGP_OUT4

Note

The **Bit Code: 111**, *both for Servo Processor 0 and Servo Processor 1, is used for switching the physical output pins to Fast General Purpose Outputs:* **FGP_OUT0** to **FGP_OUT6**. *The state of the Fast General Purpose Outputs can be read or changed using the ACSP+* **OUT(x)** *variable. The Fast General Purpose Outputs are mapped as follows:*

FGP_OUT0 is mapped to bit 16 of the **OUT(x)** variable
FGP_OUT1 is mapped to bit 17 of the **OUT(x)** variable
FGP_OUT2 is mapped to bit 18 of the **OUT(x)** variable
FGP_OUT3 is mapped to bit 19 of the **OUT(x)** variable
FGP_OUT4 is mapped to bit 20 of the **OUT(x)** variable
FGP_OUT5 is mapped to bit 21 of the **OUT(x)** variable
FGP_OUT6 is mapped to bit 22 of the **OUT(x)** variable

Comments

ASSIGNPOUTS is a blocking command in the sense that the ACSPL+ progam moves to the next line or command only after this command has been fully executed or an error is generated.

Related ACSPL+ Commands

ASSIGNPEG, **ASSIGNFINS**, **PEG_I**, **PEG_R**, **STOPPEG**

COM Library Methods

None

C Library Functions

None

2.3.4 ASSIGNFINS

Description

The **ASSIGNFINS** function is used for setting input pins assignment and mapping between **FGP_IN** signals to the bits of the **IN** variable.

IN is an integer array and stores the current state of the General Purpose digital inputs - see *SPiiPlus ACSPL+ Command & Variable Reference Guide*.

Syntax

ASSIGNFINS *axis, input_index, bit_code*

Arguments

axis	The axis index, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
input_index	0 for **IN_0**, 1 for **IN_1**, ..., 3 for **IN_9**.
bit_code	Bit code for mapping engines inputs to physical inputs according to **Table 13** and **Table 14**.

Table 13 Mapping of Engine Inputs to Physical Inputs (Servo Processor 0)

Bit Code	IN_4 MARK-1 Z	IN_3	IN_2	IN_1 MARK-1 Y	IN_0 MARK-1 X
000 (default)	None	None	None	None	None
111	FGP_IN2	Reserved	Reserved	FGP_IN1	FGP_IN0

Table 14 Mapping of Engine Inputs to Physical Inputs (Servo Processor 1)

Bit Code	IN_9	IN_8	IN_7	IN_6	IN_5 MARK-1 T
000 (default)	None	None	None	None	None
111	Reserved	Reserved	Reserved	Reserved	FGP_IN3

Note	
	The **Bit Code: 111**, *both for Servo Processor 0 and Servo Processor 1, is used for switching the physical input pins to Fast General Purpose Inputs:* **FGP_IN0** to **FGP_IN3**. *The state of the Fast General Purpose Inputs can be read or changed using the ACSP+* **IN(x)** *variable. The Fast General Purpose Inputs are mapped as follows:* FGP_IN0 is mapped to bit 16 of the **IN(x)** variable FGP_IN1 is mapped to bit 17 of the **IN(x)** variable FGP_IN2 is mapped to bit 18 of the **IN(x)** variable FGP_IN3 is mapped to bit 19 of the **IN(x)** variable *All other physical input pins are unaffected.* *Bit Codes 001-110 are not used.*

Comments

ASSIGNFINS is a blocking command in the sense that the ACSPL+ progam moves to the next line or command only after this command has been fully executed or an error is generated.

Related ACSPL+ Commands

ASSIGNPEG, **ASSIGNPOUTS**, **PEG_I**, **PEG_R**, **STOPPEG**

COM Library Methods

None

C Library Functions

None

2.3.5 DC

Description

DC (Data Collection) accumulates data of any specified standard or user-defined variable with a constant sampling rate. **DC** synchronized with motion (see Command Option **/s**) is called Axis Data Collection. **DC** not synchronized with motion is called System Data Collection.

DC terminates due to:

- **STOPDC**
- The defined **DC** array is completed

Syntax (except for DC/s)

DC*[command options] array_name, number of points, time-interval, variable_1, [variable_2...variable_N]*

Syntax for DC/s

DC*/s axis, global array, number of points, time-interval, variable_1, [variable_2...variable_N]*

Arguments

global array	The name of a global array that stores samples
number of points	Define the number of samples
time-interval	Define the time interval between each sample
variable_1	Define variable/s to be sampled. The number of rows defined in ***array_name*** must match the number of variables to be sampled

Command Options

/s	Triggers DC with the next motion command. **DC** synchronized with motion is called Axis Data Collection
/w	Create the synchronous data collection, but do not start until **GO**. Command option **/w** can only be used with the **/s**.
/c	Cyclic data collection

Comments

1. The maximum number of elements in a **DC** array is 100,000 (number of variables x number of samples ≤100,000).
2. **DC** can include up to 10 sampled variables.
3. Only one **DC** (system data collection) process can run at the same time.
4. Up to eight **DC/s** (axis data collection) processes can be simultaneously executed where each process fills a separate array.
5. **DC/c** does not self-terminate. **STOPDC** terminates cyclic data collection.
6. **DC/c** uses the collection array as a cyclic buffer and can continue to collect data indefinitely. When the array is full, each new sample overwrites the oldest sample in the array.
7. After the cyclic data collection concludes, the controller reorganizes the sample array so that the first element represents the oldest sample and the last element represents the most recent sample.
8. Variable **S_ST.#DC** = 1 when non-synchronized **DC** is active.
9. Variable **AST**<*axis*>**.#DC** = 1 when synchronized **DC** is active.

Related ACSPL+ Commands

STOPDC, **SPDC**

Related ACSPL+ Variables

S_ST, **AST**, **S_DCN**, **S_DCP**, **DCN**, **DCP**

COM Library Methods

DataCollection, StopCollect

C Library Functions

acsc_DataCollection, acsc_StopCollect

Examples

Example 1:

Cyclic Data Collection

```
GLOBAL REAL ARRAY (2)(1000)
                        !Define a real type array for two variables
                        !(rows) and 1000 sampling points (columns).
DC/C ARRAY,1000,3,FPOS0,TIME
                        !Start cyclic data collection and store 1000
                        !samples in SAMPLE_ARRAY. The time ARRAY.
                        !The time between each sampling point is 3 msec.
                        !The FPOS0 standard variable samples are stored
                        !in the first row of ARRAY, and the TIME
                        !variable values are stored in the second row.
TILL ^S_ST.#DC          !Wait until S_ST.#DC = 0 (DC collection is
                        !finished).
STOP
```

Example 2:

Motion Synchronized Data Collection

```
GLOBAL REAL ARRAY (2)(1000)
                        !Define a real type array for two variables
                        !(rows) and 1000 sampling points (columns).
DC/S 0,SAMPLE_ARRAY,1000,3,FPOS0,TIME
                        !Start cyclic data collection when motion
                        !(synchronized on axis 0) begins, and store 1000
                        !samples in SAMPLE_ARRAY. The time ARRAY. The time
                        !between each sampling point is 3 msec.
                        !The FPOS0 standard variable samples are stored in
                        !the first row of SAMPLE_ARRAY, and the TIME
                        !variable values are stored in the second row.
TILL ^AST(0).#DC        !Wait until AST.#DC = 0 (DC collection is
                        !finished).
STOP
```

2.3.6 STOPDC

Description
Immediately terminates **DC** and **SPDC**.

Syntax
STOPDC

Command Options

/s	Terminate synchronized data collection

Comments
1. **STOPDC** with an argument delays termination of **DC**.
2. **STOPDC/s** terminates synchronous **DC** initiated by **DC/s**.
3. Multiple axis specification is not allowed.

Related ACSPL+ Commands
DC, **SPDC**

Related ACSPL+ Variables
S_ST, **AST**, **S_DCN**, **S_DCP**, **DCN**, **DCP**

COM Library Methods
DataCollection, StopCollect, WaitCollectEnd

C Library Functions
acsc_DataCollection, acsc_StopCollect, acsc_WaitCollectEnd

Examples
Example 1:

```
STOPDC 50              !Collect an additional 50 samples and then
                       !terminate DC.
```

Example 2:

```
STOPDC/S 1             !Stop synchronous axis data collection for axis 1
```

2.3.7 PEG_I

Description

Position Event Generation (PEG) is a fast output, based on an axis position with sub microsecond delay. In addition to position-based pulses, the controller can also generate a specified number of time based pulses.

PEG_I is an incremental **PEG** where the controller generates a pulse at a predefined start position followed by a stream of pulses at a specified distance interval, up to a predefined end position.

Before using **PEG**, initialize the outputs using **ASSIGNPEG**.

Syntax

PEG_I *axis, width, first_point, interval, last_point*

Arguments

axis	The axis index, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
width	Width of the Pulse.
first_point	First point for the PEG generation.
interval	The distance between PEG events.
last_point	Last point for PEG generation.

Comments

1. The parameters that can be set by the command are identical to those that can be set for non-NT SPiiPlus controllers except for *time-based-pulses* and *time-based-period* the setting of which is not supported for SPiiPlus NT/DC controllers.
2. When the **PEG** pulse is activated, the voltage between the two differential PEG outputs (+) and (-) drops to -5V. When the PEG pulse is non-activated, the voltage between the two differential **PEG** outputs is 5V.
3. SPiiPlus products that provide **PEG** outputs and the general purpose digital outputs on the same pins, use **ASSIGNPEG** before **PEG** activation.
4. If the **PEG** interval is short and the time between pulses is less than one MPU cycle, additional pulses might be generated at points beyond the last **PEG** position.
5. **PEG** works correctly if the sign of the interval corresponds to the direction of motion. If the motor moves in the opposite direction to the interval sign, **PEG** pulses may be generated in unexpected positions.
6. When using a Sin-Cos encoder, **PEG** is triggered at the zero crossing of the sine-cosine waves and not at the precise interpolated position.
7. Theoretically the minimum time interval between two incremental PEG pulses is 200ns, i.e., the max frequency = 5MHz.

Related ACSPL+ Commands

PEG_R, **STOPPEG**, **ASSIGNPEG**

Related ACSPL+ Variables

AST

COM Library Methods
None

C Library Functions
None

Example
```
!!!
```

Arguments

axis	The axis index, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
width	Width of the Pulse.
mode	Output signal configuration according to **Table 15**.
first_index	Index of first entry in the array for PEG generation
last_index	Index of last entry in the array for PEG generation
POS_ARRAY	The Random Event Positions array, maximum of 256 members
OUT_ARRAY	The Outputs States array defining the four PEG output states, maximum of 256 members

Table 15 PEG Output Signal Configuration (page 1 of 2)

Bit	Signal	Description	Default Value
0	Pulse polarity	0- Output-0 positive pulse 1- Output-0 negative pulse	‘0’
1	State polarity	0- Output-0 positive state 1- Output-0 negative state	‘0’
2-3	Output type	00- Output-0 three state 01- Output-0 State 10- Output-0 Pulse 11- Output-0 Pulse&State	‘00’
4	Pulse polarity	0- Output-1 positive pulse 1- Output-1 negative pulse	‘0’
5	State polarity	0- Output-1 positive state 1- Output-1 negative state	‘0’
7-6	Output type	00- Output-1 three state 01- Output-1 State 10- Output-1 Pulse 11- Output-1 Pulse&State	‘00’
8	Pulse polarity	0- Output-2 positive pulse 1- Output-2 negative pulse	‘0’
9	State polarity	0- Output-2 positive state 1- Output-2 negative state	‘0’
11-10	Output type	00- Output-2 three state 01- Output-2 State 10- Output-2 Pulse 11- Output-2 Pulse&State	‘00’
12	Pulse polarity	0- Output-3 positive pulse 1- Output-3 negative pulse	‘0’
13	State polarity	0- Output-3 positive state 1- Output-3 negative state	‘0’

Table 15 PEG Output Signal Configuration (page 2 of 2)

Bit	Signal	Description	Default Value
15-14	Output type	00- Output-3 three state 01- Output-3 State 10- Output-3 Pulse 11- Output-3 Pulse&State	"00"

Comments

1. The parameters that can be set by the command differ from those that can be set for non-NT SPiiPlus controllers with the addition of the new *first_index* and *first_index* parameters. The setting of parameters: *time-based-pulses* and *time-based-period*, however, which can be set for non-NT SPiiPlus controllers is not supported.
2. When the **PEG** pulse is activated, the voltage between the two differential **PEG** outputs (+) and (-) drops to -5V. When the **PEG** pulse is de-activated, the voltage between the two differential **PEG** outputs is 5V.
3. In **PEG_R**, the number of position-based pulses is limited to eight pulses per controller cycle and the minimum time interval must be >200nsec.
4. When a **PEG STATE** is activated, the voltage between the two differential **PEG** outputs (+) and (-) becomes 5V. When the **PEG STATE** is de-activated, the voltage between the two differential PEG outputs is -5V.
5. The last three arguments are optional. If array_state is omitted, the controller generates the PEG pulses at each position but does not change the state of any output. If time-based-pulses and time-based-period are omitted, the controller does not generate time based pulses.
6. When using a Sin-Cos encoder, **PEG** is triggered at the zero crossing of the sine-cosine waves and not at the precise interpolated position.
7. For SPiiPlus products that provide **PEG** outputs and general purpose digital outputs on same pins, use **ASSIGNPEG** before **PEG** activation.

Related ACSPL+ Commands

PEG_I, **STOPPEG**, **ASSIGNPEG**

Related ACSPL+ Variables

AST

COM Library Methods

None

C Library Functions

None

Example

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!PEG pulses are fired on axes 0,1,2 according to encoder of axis 0 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
GLOBAL AXIS0
GLOBAL AXIS1
GLOBAL AXIS2

GLOBAL REAL ARR(16) !Definition of a 16 point position array
ARR(0)=100;ARR(1)=150;ARR(2)= 200;ARR(3)=250;
```

```
ARR(4)=300;ARR(5)=350;ARR(6)=400;ARR(7)= 450;
ARR(8)=500;ARR(9)=550;ARR(10)= 600;ARR(11)=650;
ARR(12)=700;ARR(13)=750;ARR(14)=800;ARR(15)=1000;

GLOBAL INT STAT(16) !Definition of 16 point state array. The values define
PEGx_OUTy
STAT(0)=0b0001;STAT(1)=0b0100;STAT(2)=0b1111;STAT(3)=0b1110;
STAT(4)=0b0101;STAT(5)=0b0001;STAT(6)=0b0100;STAT(7)=0b1111;
STAT(8)=0b1110;STAT(9)=0b0101;STAT(10)=0b0001;STAT(11)=0b0100;
STAT(12)=0b1111;STAT(13)=0b0001;STAT(14)=0b0100;STAT(15)=0b1111;

AXIS0=0
AXIS1=1
AXIS2=2

set FPOS(AXIS0)=0
set FPOS(AXIS1)=0
set FPOS(AXIS2)=0

assignpeg AXIS0, 0b100, 0b0

assignpouts AXIS0, 0, 0b000 !Assign bit code 000 from Table 11 to AXIS0, at Pin0
assignpouts AXIS1, 1, 0b000
assignpouts AXIS2, 2, 0b001

ST:
peg_r AXIS0,0.5,0x4444,0,15,ARR,STAT !Activate random PEG for axis 0
       peg_r AXIS1,0.5,0x4444,0,15,ARR,STAT
       peg_r AXIS2,0.5,0x4444,0,15,ARR,STAT

TILL AST(AXIS0).#PEGREADY
!Wait till command executes and configuration is set, in order to ensure proper
!PEG engine's execution  prior to start of movement
       TILL AST(AXIS1).#PEGREADY
       TILL AST(AXIS2).#PEGREADY

       ptp/e AXIS0,5000
       stoppeg AXIS0 ! Prevent PEG pulses' firing on the 'way back'
       ptp/e AXIS0,0
goto ST

stop
```

2.3.9 STARTPEG

Description

The **STARTPEG** command initiates the PEG process on the specified axis. The command is used in both the Incremental and Random PEG modes.

Syntax

STARTPEG *axis*

Arguments

axis	The axis index, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.

Comments

STARTPEG is a blocking command in the sense that the ACSPL+ progam moves to the next line or command only after this command has been fully executed or an error is generated.

If **STOPPEG** has been issued before the last PEG position, you have to use **STARTPEG** to resume PEG engine firings from the current point.

Related ACSPL+ Commands

PEG_I, **PEG_R**, **STOPPEG**, **ASSIGNPEG**

Related ACSPL+ Variables

AST

COM Library Methods

None

C Library Functions

None

2.3.10 READ

Description

Reads a file from the controller's nonvolatile (flash) memory to a user defined array. The file must exist in the nonvolatile memory by previously writing it using the **WRITE** command.

Syntax

READ *array,filename*

Arguments

array	User defined array to which the data will be imported
filename	Non-volatile memory file name

Comments

1. The ***filename*** must not include the file name extension.
2. Declare the user array name in the buffer where the command is executed.
3. If **READ** is executed from the **Communication Terminal** window as an immediate command, ***array*** must specify a global array.
4. **READ** can not be used with the SPiiPlus Simulator.

Related ACSPL+ Commands

WRITE

COM Library Methods

Read

C Library Functions

acsc_Read

Example

```
GLOBAL REAL ARRAY(5)     !Define a real global array
READ ARRAY,FILENAME      !Read the defined file to the array
```

2.3.11 SETPROTECTION

Description
SETPROTECTION sets any ACSPL+ variable as read-only when the controller is in Protected mode. Enable or disable protected mode with the **Communication Terminal** commands: **PROTECT** and **UNPROTECT**, respectively. **SETPROTECTION** takes effect only after the controller is set to the Protected mode.

Syntax
SETPROTECTION *standard_variable*

Arguments

standard_variable	Any standard ACSPL+ variable.

Comments

1. **PFLAGS**<*buffer_number*>**.#NOEDIT** disabled changing buffer in protected mode. The bit can be defined separately for each buffer.
2. **PFLAGS**<*buffer_number*>**.#NOVIEW** disables viewing the buffer in protected mode.The bit can be defined separately for each buffer.
3. The controller accepts **#RESET** in protected mode. This way, the user can delete a protected application, and reset the controller to the factory-default state, even if the password is unknown. Access **#RESET** from the SPiiPlus MMI menu command **Application.**

Related Variables
CFG

Example

```
SETPROTECTION VEL=1
```

2.3.12 SPDC

Description
SPDC (Servo Processor Data Collection) performs fast data collection and accumulates data about the specified SP variable with a constant maximum sampling rate of 20kHz. A typical use for **SPDC** is for collecting position error (**PE**) and feedback position (**FPOS**) data at the fast SP rate.

The SP value is different from the MPU value. The SP always uses counts and not units. The SP position value is not affected by SET FOPS command. An offset is added at the MPU level only. The formula (that you can find in our manuals) is:

FPOS = FP*EFAC + EOFFS

where FPOS is the MPU variable and FP is the SP calculated value.

SPDC terminates due to:

- ❑ **STOPDC**
- ❑ After accumulating the defined number of samples

Syntax

SPDC *user_array, number_of_samples, sampling_interval, SP_serial_number, SP_memory_address, shift*

Arguments

user_array	Integer array only, up to 30k
number_of_samples	The maximum value is up to the size of the array
sampling_interval	0.05msec (20kHz) to 1msec (when **CTIME**=1)
SP_serial_number	The SP serial number.
SPii_memory_address	Memory address for the SP variable, e.g., 0x0b7 for PE(0).
shift	Specifies scaling for the samples as a number of right shifts. The controller collects 12-bit values by default, if higher bit values are required (16-bits, for example), then ***shift*** must equal the number of zeros added before the **least** significant bit. For example, a 16-bit value requires ***shift*** = 4.

Note	
	*Since memory addresses may vary between SPiiPlus products and revisions, it is highly recommended to define a variable to represent **SP_memory_address** as the return value of **GETSPA**. **SPDC** can then use this variable in any SPiiPlus product or revision.*

Related ACSPL+ Commands

STOPDC

Comments

1. Only one **SPDC** command can run at the same time.
2. **Table 16** lists the most commonly used SP variables for **SPDC**.

Table 16 Common Monitored SPDC Variables (page 1 of 2)

Variable	Axis	SP Variable	Shift
Position Error	0,1,2,3	PE(0)	0
Position Error	4,5,6,7	PE(4)	0
Feedback Position	0,1,2,3	CP_L(0)	Per case

Table 16 Common Monitored SPDC Variables (page 2 of 2)

Variable	Axis	SP Variable	Shift
Feedback Position	4,5,6,7	CP_L(4)	Per case
Feedback Velocity	0,1,2,3	VEL(0)	Per case
Feedback Velocity	4,5,6,7	VEL(4)	Per case
Sin Analog Input	0,1,2,3	ADC0_X_MUL_A_PWM2	2
Sin Analog Input	4,5,6,7	ADC2_A_MUL_A_PWM4	2
Cos Analog Input	0,1,2,3	ADC1_X_MUL_B_PWM3	2
Cos Analog Input	4,5,6,7	ADC3_A_MUL_B_PWM5	2
Phase A Current	0,1,2,3	IA(0)	2
Phase A Current	4,5,6,7	IA(4)	2
Current Command	0,1,2,3	IA_COMMAND(0)	2
Current Command	4,5,6,7	IA_COMMAND(4)	2

Example

```
! ------------ Declare data arrays ----------------------
GLOBAL INT DATA(15000)  !Declare global array of integer of size 15000
REAL PAR_ADDRESS
PAR_ADDRESS=GETSPA(0,"PE(0)")
! ------------ Define motion parameters------------------------------
VEL(0) = 5000
ACC(0) = 100000
DEC(0) = 100000
JERK(0) =2e6
!--------------Run motion and do fast data collection ----------------
SET FPOS(0)=0
ENABLE X
SPDC DATA,15000,0.05,0,PAR_ADDRESS,0
PTP/e 0, 1000
PTP/e 0, -1000
PTP/e 0, 1000
STOP
!Use the following if you need to convert the data to one column to
!export to Excel (otherwise you can collect 30000 points by SPDC above)
!Convert the array data from one row to one column to fit to export to Excel.
NT DATA1(15000)(1)
INT J; J=0
LOOP 10000;DATA1(J)(0)=DATA(J);J=J+1;END
STOP
```

After **SPDC** is finished, the content of **USER** contains only 20 members since data collection was stopped after 1 msec where a sample was collected every 50 μsec.

2.3.13 STOPPEG

Description
STOPPEG terminates the **PEG** immediately. The function is used with both Incremental and Random PEG modes

Syntax
STOPPEG *axis*

Arguments

axis	Specified axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.

Related ACSPL+ Commands
PEG_I, **PEG_R**

COM Library Methods
None

C Library Functions
None

Example

```
PEG_I/S  0, 0.003, 1000, 1000, 3000, 2, 0.01
PTP 0, 3000             !The program initiates synchronous PEG, with PTP
                        !motion on axis 0.
WAIT 2                  !Two milliseconds after motion starts, PEG
                        !will be terminated by STOPPEG.
STOPPEG 0
```

2.3.14 WRITE

Description
Writes an array (any system or user-defined variable) to a file in the controller's nonvolatile (flash) memory.

Syntax

WRITE *user-array*, *filename*

Arguments

user-array	User defined array from which the data will be imported
filename	Non-volatile memory file name

Comments

1. The nonvolatile memory **filename** must not include an extension.
2. The **user-array** name must be declared in the buffer where the command is executed.
3. If **WRITE** is executed from the **Terminal** window as an Immediate command, specify the **user-array** name as a global array.
4. If **filename** does not exist, it is created. If the file already exists, it is over written.
5. **WRITE** cannot be used with the SpiiPlus Simulator

Related ACSPL+ Commands

READ

COM Library Methods

Write

C Library Functions

acsc_Write

2.4 Motion Commands

The Motion commands are:

Command	Action
ARC1	Adds a circle segment to **MSEG...ENDS** motion.
ARC2	Adds a circle segment to **MSEG...ENDS** motion.
JOG	Creates a jog motion.
LINE	Adds a linear segment to **MSEG...ENDS** motion.
MASTER	Defines a formula for calculating **MPOS**.
MPOINT	Adds a set of points to **MPTP...ENDS**, **PATH...ENDS** **or** **PVSPLINE...ENDS** motion.
MPTP...ENDS	Creates a multipoint motion.
MSEG...ENDS	Creates a segmented motion.
PATH...ENDS	Creates an arbitrary path motion with linear interpolation between the specified points.
POINT	Adds a point to **MPTP...ENDS**, **PATH...ENDS,** or **PVSPLINE...ENDS** motion.
PROJECTION	An expansion command to the **MSEG...ENDS** set of commands, that allows the controller to perform a three dimensional segmented motion such as creating arcs and lines on a user-defined plane.
PTP	Creates a point-to-point motion.

Command	Action
PVSPLINE...ENDS	Creates an arbitrary path motion with spline interpolation between the specified points.
SLAVE	Creates a master-slave motion.
STOPPER	Adds a segment separator to **MSEG...ENDS** motion.
TRACK	Creates tracking motion.

2.4.1 ARC1

Description

ARC1 must be initialized with **MSEG...ENDS**. Use **ARC1** to specify the center point and final point coordinates of an arc and the direction of rotation. Direction is designated by a plus sign (+) or (–) for clockwise or counterclockwise rotation depending on the encoders' connections.

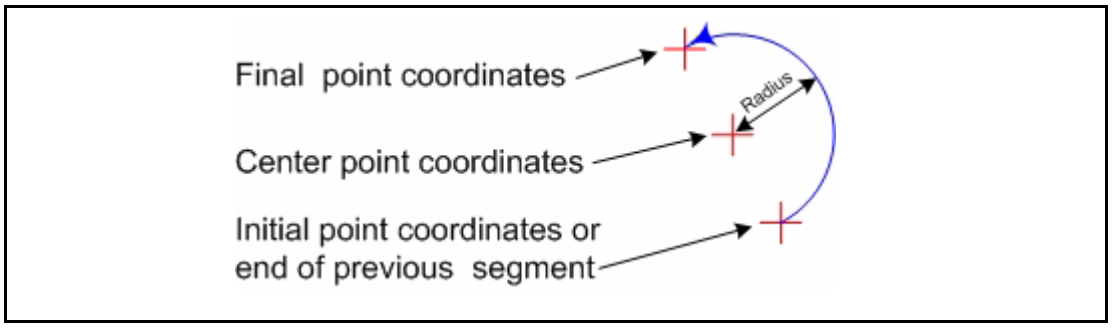


Figure 6 ARC1 Coordinate Specification

Syntax

ARC1 *axis_list, center-point, final-point, direction[,user-specified velocity]*

Arguments

axis_list	List of axes involved, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1. The **ARC1** axis_list can involve two or more axes, see **PROJECTION**.
center-point	Center point coordinates.
final point	Last point.
direction	Use + for motion in the direction of increasing encoder counts, or - for motion in the direction decreasing encoder counts.
user-specified velocity	If **MSEG** command option **/v** is used, the user-specified velocity must be the last parameter in the **ARC1** syntax.

Comments

1. **ARC1** and **ARC2** differ only by the required arguments. **ARC1** requires the coordinates of the center point, final point, and the direction of rotation. **ARC2** requires the coordinates of the center point and the rotation angle in radians. Each command produces the same result, so selection of either **ARC1** or **ARC2** depends on the available data.

2. A single **ARC1** command can not create a complete circle because the start point and end point of the motion can not be the same. Use two **ARC1** commands, or use **ARC2**.

Related ACSPL+ Commands
MSEG...ENDS, **ARC2**, **LINE**, **STOPPER**, **PROJECTION**

COM Library Methods
Arc1, ExtArc1

C Library Functions
acsc_Arc1

Example
See **MSEG...ENDS**.

2.4.2 ARC2

Description
ARC2 must be initialized with **MSEG...ENDS**. Use **ARC2** to specify the center point and rotation angle in radians of an arc segment. Designate direction by positive or negative rotation angle, depending on the encoders' connections.

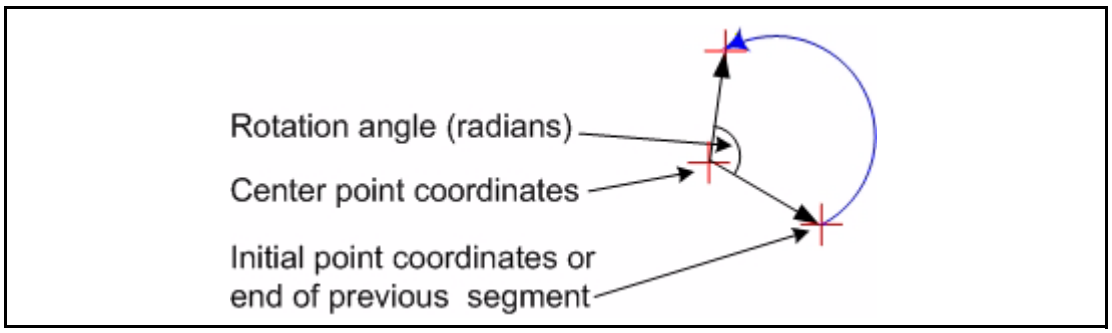


Figure 7 ARC2 Center Point and Rotation Angle Specification

Syntax
ARC2 *axis_list, center-point, rotation-angle and direction [,user-specified velocity]*

Arguments

axis_list	List of axes involved, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1. The **ARC2** ***axis_list*** can involve two or more axes, see **PROJECTION**.
center-point	center point coordinates
final point	last point
rotation angle and direction	Rotation is in radians. Use + for motion in the direction of increasing encoder counts, or - for motion in the direction decreasing encoder counts
user-specified velocity	If **MSEG** command option **/v** is used, the user-specified velocity must be the last parameter in the **ARC2** syntax.

Comments

ARC1 and **ARC2** differ only by the required arguments. **ARC1** requires the coordinates of the center point, final point, and the direction of rotation. **ARC2** requires the coordinates of the center point and the rotation angle. Each command produces the same result, so selection of either **ARC1** or **ARC2** depends on the available data.

Related ACSPL+ Commands

MSEG...ENDS, **ARC1**, **LINE**, **STOPPER**, **PROJECTION**

COM Library Methods

Arc2, ExtArc2

C Library Functions

acsc_Arc2, acsc_ExtArc2

Example

See **MSEG...ENDS**.

2.4.3 JOG

Description

JOG creates a motion with constant velocity and no defined end point.

JOG motion terminates by:

- **HALT**, **KILL**, **BREAK**, **DISABLE**
- Execution of any other motion command for the same axis
- Limit switch activation
- Any fault activation that disables the drive or kills the motion

Syntax

JOG*[/command-options] axis_list [,user-specified-velocity][,user-specified-direction]*

Arguments

axis_list	List of axes, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
user-specified-velocity	Optional parameter used if the **/v** command option is specified
user-specified-direction	Optional parameter. Motion direction is designated by a plus sign (+) for increasing feedback counts or (–) or decreasing feedback counts. If no operator is used, motion is in the direction of increasing encoder counts.

Command Options

/v	Use the specified velocity instead of the default velocity (**VEL**).
/w	Create the motion, but do not start until **GO**.

Related ACSPL+ Commands
GO, **HALT**, **KILL**, **BREAK**, **IMM**

Related ACSPL+ Variables
AST, **MST**, **MERR**, **VEL**, **ACC**, **JERK**, **FPOS**, **RPOS**, **GPHASE**

COM Library Methods
Jog, JogM

C Library Functions
acsc_jog, acsc_jogM

Examples
Example 1:

```
JOG/v 0, 500            !Jog 0 axis with VEL 500
```

Example 2:

```
JOG (0,1,2), -++        !Jog axes 0, 1, and 2 where axis 0 jogs in the
                        !negative direction and axes 1 and 2 jog in the
                        !positive direction.
```

2.4.4 LINE

Description
LINE must be initialized with **MSEG...ENDS**. **LINE** adds a linear segment to a segmented motion.

Syntax
LINE *axis_list, final-position1, final-position2[,user-specified velocity]*

Arguments

axis_list	List of axes, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
final position1	First coordinate
final position2	Second coordinate
user specified velocity	Optional user-specified velocity if **LINE** was initiated by **MSEG/v**.

Comments

1. **ENDS** informs the controller that no more segments will be specified. If **ENDS** is omitted, motion stops at the last point of the sequence and waits for the next point.
2. If the **LINE** axis_list involves two or more axes, see **PROJECTION**.

Related ACSPL+ Commands

MSEG...ENDS, **ARC1**, **ARC2**, **STOPPER**, **PROJECTION**

COM Library Methods

See: Points and Segments Manipulation

C Library Functions

See: Points and Segments Manipulation

Example

See **MSEG...ENDS**.

2.4.5 MASTER

Description

MASTER defines master-slave motion by creating a dependency between an axis position or velocity to a variable and/or an expression. **MASTER** always follows the (axis) **MPOS** variable. **SLAVE** initiates the motion defined by **MASTER** and must follow **MASTER**.

Note	
	***Velocity Lock** is the default state for **MASTER**. Initiate **MASTER** with **SLAVE**/**p** for position lock.*

The following actions terminate the master-slave dependency, however, these actions do not necessarily terminate the motion:

- **KILL** or **HALT** to the slave axis.
- **DISABLE** to either axis or both axes.
- Setting the logical dependence between master and slaves axes to zero. For example, **MASTER MPOS(0) = 0**.

Syntax

MASTER *axis_MPOS=formula*

Arguments

axis_MPOS	Define master value for axis
=	Assignment operator
formula	A variable and/or expression

Related ACSPL+ Commands
SLAVE, **GO**, **HALT**, **KILL**, **ENABLE/ENABLEALL**, **DISABLE**, **MAP**

Related ACSPL+ Variables
AST, **XSACC**, **SYNV**, **MFF**

COM Library Methods
SetMaster, Slave, SlaveStalled

C Library Functions
acsc_SetMaster, acsc_Slave, acsc_SlaveStalled

Examples
Example 1:

```
MASTER MPOS(0)=5*RVEL(1)
                        !Creates a master-slave dependency where
                        !the 0 axis velocity is slaved to five times
                        !the 1 axis reference velocity (RVEL)
SLAVE 0                 !Initiates the 0 axis motion
```

Example 2:

Figure 8 illustrates **SLAVE/pt** used in the following syntax example.

```
MASTER MPOS(0)=FPOS(1)+MARK (1)
                        !Create a master slave dependency, where
                        !axis 0 is the slave. The master axis
                        !is 1, and the FPOS (feedback position)
                        !expression plus the MARK position
                        !are the references for the slaved axis 0.
SLAVE/pt 0, -500, 500
                        !SLAVE command with position lock
                        !and specified boundary switches,
                        !for axis 0. -500 is the left
                        !position boundary, and 500 is the
                        !right position boundary.
```

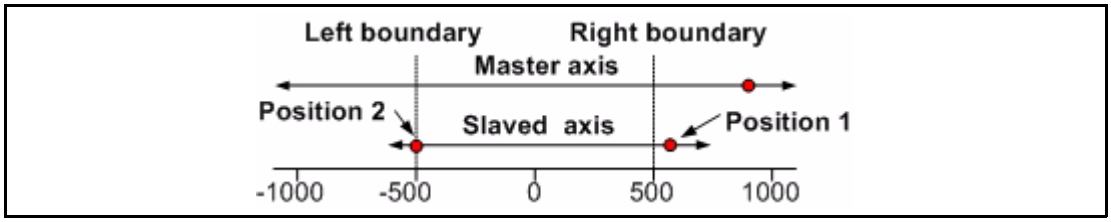


Figure 8 SLAVE /pt Illustration

In **Figure 8**, Position 1 is outside of the defined boundary and the master-slave dependency is stalled. Position 2 is within the defined boundary and the master-slave dependency is active.

2.4.6 MPOINT

Description

MPOINT specifies an array of destination points used by **MPTP...ENDS**, **PATH...ENDS** or **PVSPLINE...ENDS** motion commands. An **MPOINT** array must conclude with **ENDS**.

Syntax

MPOINT *axis_list, array-name (number of rows,number of points)*

Arguments

axis_list	List of axes involved. Axes must be in the same order as defined in **MPTP**, **PATH** or **PVSPLINE**.
array (number of rows,number of points)	The **MPOINT** array must be defined as **Real**. The number of rows is the first element and the number of columns is the second element. A three-axis, five-point array is defined as follows: **real array (3)(5)**. Each row defines the values for the points for each axes, and depending on command preceding **MPOINT**, the defined velocities, and the defined times. The array must contain, at least, the same number of columns declared in the array as the number of defined points. Extra array columns are ignored. See **Array Structures** for a description of various array configurations.

Related ACSPL+ Commands

MPTP...ENDS, **POINT**, **PATH...ENDS**, **PVSPLINE...ENDS**

COM Library Methods

MultiPoint, MultiPointM, Spline, SplineM, AddPVPoint, AddPVPointM, AddPVTPoint, AddPVTPointM

C Library Functions

acsc_MultiPoint, acsc_MultiPointM, acsc_Spline, acsc_SplineM, acsc_AddPVPoint, acsc_AddPVPointM, acsc_AddPVTPoint, acsc_AddPVTPointM

Array Structures

There are different point array structures depending on the **MPOINT** enabling command and command options. These structures are described below.

1. A five-point, three-axis **MPOINT** array enabled by **MPTP** or **PATH** without command options appears as follows:

ARRAY (3)(5)

	Point 1	**Point 2**	**Point 3**	**Point 4**	**Point 5**
0 Axis	1000	2000	3000	4000	5000
1 Axis	5000	4000	3000	2000	1000
5 Axis	0	2000	4000	6000	8000

See Example 1.

2. If **MPOINT** follows **MPTP**/v, the point array must include an additional row to specify the transition velocity from the previous point to the current point and appears as follows:

ARRAY (4)(5)

	Point 1	**Point 2**	**Point 3**	**Point 4**	**Point 5**
0 Axis	1000	2000	3000	4000	5000
1 Axis	5000	4000	3000	2000	1000
5 Axis	0	2000	4000	6000	8000
Velocity	6000	7000	5000	4000	4000

See Example 2.

3. If **MPOINT** follows **PATH/t**, the point array must include an additional row to specify the time interval between the previous point and the current point and appears as follows:

ARRAY (4)(5)

	Point 1	**Point 2**	**Point 3**	**Point 4**	**Point 5**
0 Axis	1000	2000	3000	4000	5000
1 Axis	5000	4000	3000	2000	1000
5 Axis	0	2000	4000	6000	8000
Time	250	250	250	250	250

Time is specified in milliseconds.

4. If **MPOINT** follows **PVSPLINE** without command option **/t**, the array must include two sets of rows for each axis:
   - The first set defines the destination points
   - The second set defines the velocity at the destination points

   For example:

ARRAY (6)(5)

	Point 1	Point 2	Point 3	Point 4	Point 5
0 Axis	1000	2000	3000	4000	5000
1 Axis	5000	4000	3000	2000	1000
5 Axis	0	2000	4000	6000	8000
0 Axis VEL	5000	5000	5000	5000	5000
1 Axis VEL	2500	2500	2500	2500	2500
5 Axis VEL	3000	3000	3000	3000	3000

5. If **MPOINT** follows **PVSPLINE/t**, the array must include an additional column that specifies the time interval between the previous point and the current point. **Time** is in milliseconds. For example:

ARRAY (7)(5)

	Point 1	Point 2	Point 3	Point 4	Point 5
0 Axis	1000	2000	3000	4000	5000
1 Axis	5000	4000	3000	2000	1000
5 Axis	0	2000	4000	6000	8000
0 Axis VEL	5000	5000	5000	5000	5000
1 Axis VEL	2500	2500	2500	2500	2500
5 Axis VEL	3000	3000	3000	3000	3000
Time	500	500	500	500	500

See Example 3.

6. If **MPTP/r** enables **MPOINT**, the array points are relative.

Examples

Example 1:

Illustrating sample code based on the **ARRAY (3)(5)** structure.

```
REAL ARRAY(3)(5)          !Defines ARRAY with three rows and
                          !five columns as real.
INT NUMBER_of_POINTS      !Defines NUMBER_of _POINTS as an integer variable.
! ------------ Fill the ARRAY array----------------------
ARRAY(0)(0)=1000;ARRAY(0)(1)=2000;ARRAY(0)(2)=3000;ARRAY(0)(3)=4000;
ARRAY (0)(4)=5000;ARRAY(1)(0)=5000;ARRAY(1)(1)=4000;ARRAY(1)(2)=3000;
ARRAY (1)(3)=2000;ARRAY(1)(4)=1000;ARRAY(2)(0)=0;ARRAY(2)(1)=2000;
ARRAY (2)(2)=4000;ARRAY (2)(3)=6000;ARRAY(2)(4)=8000
NUMBER_of_POINTS=5        !Set NUMBER_of_POINTS
ENABLE (0,1,5)            !Enable axes 0, 1 and 5
```

```
PATH (0,1,5), 1          !PATH initiates simultaneous motion for axes 0, 1,
                         !and 5. The time interval is one millisecond between
                         !each destination point.
MPOINT (0,1,5), ARRAY, NUMBER_of_POINTS
                         !Define an MPOINT array for axes 0, 1 and 5 where
                         !ARRAY and NUMBER_of_POINTS are called
ENDS (0,1,5)             !Concludes MPOINT
STOP                     !End program
```

Example 2:

Illustrating sample code based on the **ARRAY (4)(5)** structure.

```
REAL ARRAY(4)(5)         !Defines ARRAY with four rows and
                         !five columns as real.
INT NUMBER_of_POINTS     !Defines NUMBER_of _POINTS as an integer
                         !variable.
! ------------ Fill the ARRAY array----------------------
ARRAY(0)(0)=1000;ARRAY(0)(1)=2000;ARRAY(0)(2)=3000;ARRAY(0)(3)=4000;
ARRAY (0)(4)=5000;ARRAY(1)(0)=5000;ARRAY(1)(1)=4000;ARRAY(1)(2)=3000;
ARRAY (1)(3)=2000;ARRAY(1)(4)=1000;ARRAY(2)(0)=0;ARRAY(2)(1)=2000;
ARRAY(2)(2)=4000;ARRAY(2)(3)=6000;ARRAY(2)(4)=8000;ARRAY(3)(0)=6000;
ARRAY(3)(1)=7000;ARRAY(3)(2)=5000;ARRAY(3)(3)=4000;ARRAY(3)(4)=4000
NUMBER_of_POINTS=5       !Set NUMBER_of_POINTS
ENABLE (0,1,5)           !Enable axes 0, 1 and 5
PATH (0,1,5), 1          !PATH initiates simultaneous motion for axes
                         !0, 1 and 5. The time interval is one millisecond
                         !between each destination point.
MPOINT (0,1,5), ARRAY, NUMBER_of_POINTS
                         !Define an MPOINT array for axes 0, 1 and 5
                         !where ARRAY and NUMBER_of_POINTS are called.
ENDS (0,1,5)             !Concludes MPOINT
STOP                     !End program
```

Example 3:

Illustrating sample code based on the **ARRAY (7)(5)** structure.

```
REAL ARRAY(7)(5)          !Defines ARRAY with seven rows and
                          !five columns as real.
INT NUMBER_of_POINTS      !Defines NUMBER_of _POINTS as an integer variable.
! ------------ Fill the ARRAY array----------------------
ARRAY(0)(0)=1000;ARRAY(0)(1)=2000;ARRAY(0)(2)=3000;ARRAY(0)(3)=4000;
ARRAY (0)(4)=5000;ARRAY(1)(0)=5000;ARRAY(1)(1)=4000;ARRAY(1)(2)=3000;
ARRAY (1)(3)=2000;ARRAY(1)(4)=1000;ARRAY(2)(0)=0;ARRAY(2)(1)=2000;
ARRAY(2)(2)=4000;ARRAY(2)(3)=6000;ARRAY(2)(4)=8000;ARRAY(3)(0)=5000;
ARRAY(3)(1)=5000;ARRAY(3)(2)=5000;ARRAY(3)(3)=5000;ARRAY(3)(4)=5000;
ARRAY(4)(0)=2500;ARRAY(4)(1)=2500;ARRAY(4)(2)=2500;ARRAY(4)(3)=2500;
ARRAY(4)(4)=2500;ARRAY(5)(0)=3000;ARRAY(5)(1)=3000;ARRAY(5)(2)=3000;
ARRAY(5)(3)=3000;ARRAY(5)(4)=3000;ARRAY(6)(0)=500;ARRAY(6)(1)=500;
ARRAY(6)(2)=500;ARRAY(6)(3)=500;ARRAY(6)(4)=500
NUMBER_of_POINTS=5        !Set NUMBER_of_POINTS
ENABLE (0,1,5)            !Enable axes 0, 1 and 5
PVSPLINE/t (0,1,5)        !PVSPLINE/t initiates simultaneous motion for axes
                          !0, 1 and 5 with a user-defined time interval
                          !between each point.
MPOINT (0,1,5), ARRAY, NUMBER_of_POINTS
                          !Define an MPOINT array for axes 0, 1 and 5
                          !where ARRAY and NUMBER_of_POINTS are called.
ENDS (0,1,5)              !Concludes MPOINT
STOP                      !End program
```

2.4.7 MPTP...ENDS

Description

MPTP initiates multipoint sequential positioning to a set of points. **MPTP** by itself does not specify any point, however dwell time at each point can be optionally specified. Points are specified by the **POINT** or **MPOINT** commands that follow **MPTP**.

In single axis motion, **MPTP** generates sequential motion between the defined array points, where at the end of each segment, **RVEL** = 0, as if each segment was defined by a separate **PTP** command.

In group motion, where more than one axis is declared, the first axis in the *axis_list* is the leading axis. The motion parameters of the leading axis become the default motion parameters for all axes in the group. Motion on all axes in a group start and conclude at the same time. **MPTP** generates sequential motion between the defined array points, where at the end of each set of points, **RVEL** = 0, as if each motion was defined by a separate **PTP** command.

Transition to the next motion in the motion queue, if it exists, will not occur until **ENDS** executes.

MPTP terminates with:

- **HALT**, **KILL**, or **BREAK**
- Any fault activation that disables the drive or kills the motion
- **DISABLE** by the user

Syntax
MPTP*[/command options] axis_list[,dwell]*

Arguments

axis_list	List of axes, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
dwell	Dwell is an optional argument, expressed in milliseconds.

Command Options

/w	Create the motion, but do not start until the **GO** command.
/v	Use the specified velocity instead of the default velocity (**VEL**).
/c	Use the point sequence as a cyclic array
/r	Coordinates of each point are relative to the previous point's coordinates.

Comments
1. **MPTP** motion starts only after the first point is specified
2. **MPTP** motion with all of its points is considered as one motion command in the motion queue.
3. An **MPOINT** array declaration used in **MPTP** must be defined as Real.
4. **MPTP/c** does not end automatically. Use **HALT**, **KILL**, or **BREAK** to stop cyclic motion.

Related ACSPL+ Commands
GO, **HALT**, **KILL**, **BREAK**, **IMM**, **MPOINT**, **POINT**

Related ACSPL+ Variables
ACC, **DEC**, **JERK**, **VEL**

COM Library Methods
ToPoint, ToPointM

C Library Functions
acsc_ToPoint, acscTopointM, acsc_ExtToPoint, acsc_ExtToPointM

Examples
Example 1:

In the following example, ***dwell*** is not required, therefore the comma and the second argument following 1000 are omitted.

```
MPTP X, 1000              !Create a multipoint motion of the X axis
                          !with a dwell of 1000 msec at each point.
```

Example 2:

```
REAL ARRAY1 (4)           !Define an array called ARRAY1 with four members
                          !as REAL
! ------------ Fill the ARRAY1 array----------------------
ARRAY1(0)=0;ARRAY1(1)=1000;ARRAY1(2)=0;ARRAY1(3)=1000
MPTP X, 500               !MPTP motion for X axis, dwell 500 msec at each point
POINT X, 2000             !First point
MPOINT X, ARRAY1,3        !Use three points from ARRAY1
POINT X, 3000             !Second point
POINT X, 5000             !Third point
ENDS X                    !Ends the point sequence for X axis
```

Figure 9 illustrates the above code for a single-axis motion initiated by **MPTP**. Note that not all of the members defined in ARRAY1 necessarily need to be used by **MPOINT**. In this example, only three of the defined members are called.

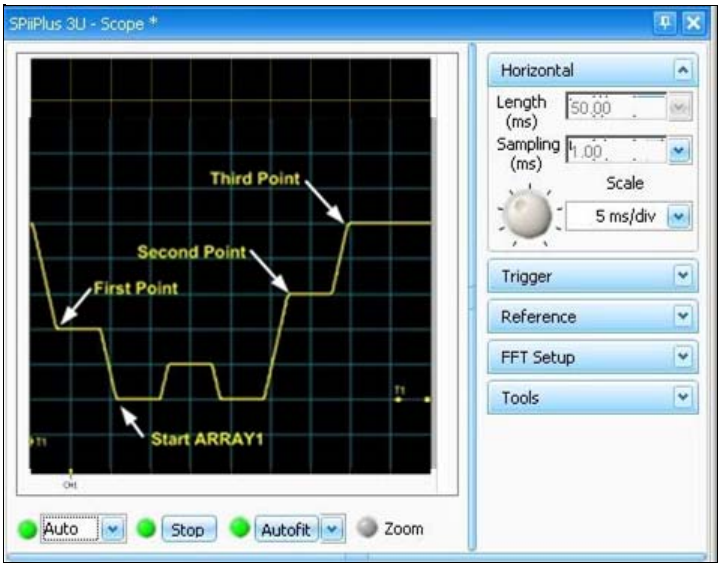


Figure 9 Single-Axis Motion Using MPTP

Example 3:

```
MPTP/v XY                !Create multipoint motion in group XY with
                         !user defined velocity and no dwell.
POINT XY,10000,0,5000    !Move to first point at velocity 5000
POINT XY,1000,1000,5000  !Add second point at velocity 5000
POINT XY,2000,1000,5000  !Add third point at velocity 5000
POINT XY,2000,0,100      !Add fourth point at velocity 100
ENDS XY                  !End the point sequence for XY
```

Figure 10 illustrates the above code for two-axis group motion initiated by **MPTP/v**.

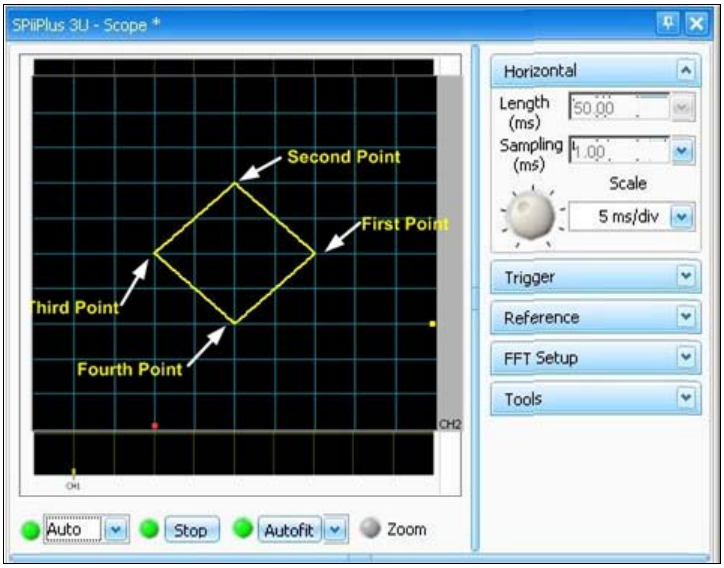


Figure 10 Two-Axis Group Motion Using MPTP/v

2.4.8 MSEG...ENDS

Description

MSEG initiates two-axis segmented motion. **MSEG** itself does not specify a line or arc segment. Motion starts only after the first segment is specified with a motion segment command.

Segmented motion moves axes along a continuous path where the path is defined as a sequence of line and arc segments on a plane.

Use the following commands to define segmented motion:

- **ARC1** - adds an arc segment to a segmented motion and specifies the coordinates of center point, coordinates of the final point, and the direction of rotation
- **ARC2** - Adds an arc segment to a segmented motion and specifies the coordinates of center point, rotation angle and direction.
- **ENDS**- terminates the point sequence

- **LINE** - adds a linear segment to a segmented motion.
- **PROJECTION** - sets a projection array for a segmented motion.
- **STOPPER** - provides a smooth transition between two segments of segmented motion.

MSEG motion terminates with:

- **HALT**, **KILL**, or **BREAK**
- Any fault activation that disables the drive or kills the motion
- **DISABLE** by the user

Syntax

MSEG*[/command options] axis_list, initial-position-axis1,initial-position-axis2*

Arguments

axis_list	Axes list, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
initial-position axis1	Start coordinate for the first axis
initial-position-axis2	Start coordinate for the second axis

Command Options

/w	Create the motion, but do not start until **GO**.
/v	Use the specified velocity instead of the default velocity (**VEL**).
/c	Use the **MSEG** segment sequence as a cyclic array.

Comments

1. Use command option /c to create cyclic motion where the final point of the last segment becomes the first point of the next motion cycle. **MSEG/c** does not automatically finish. Use **HALT**, **KILL**, or **BREAK** to end cyclic motion.
2. If command option /v is used, specify the user-defined velocity in each instance of **LINE**, **ARC1**, or **ARC2**.
3. The MSEG command uses a motion que buffer of up to 50 segments.

Related ACSPL+ Commands

GO, **HALT**, **KILL**, **BREAK**, **IMM**

Related ACSPL+ Variables

ACC, **DEC**, **JERK**, **VEL**

COM Library Methods

For **MSEG**: Segment, Line1, ExtLine, Arc1, ExtArc1, Arc2, ExtArc2, Stopper, Projection

For **ENDS**: FinalPoint

C Library Functions

For **MSEG**: acsc_Segment, acsc_Arc1, acsc_Arc2, acsc_ExtLine, acsc_ExtArc1, acsc_ExtArc2, acsc_Projection, acsc_EndSequenceM, acsc_Stopper

For **ENDS**: ascs_FinalPoint

Example

```
MSEG (0,1), 1000, 1000   !MSEG initiates segmented motion for the 0 and 1 axes
                         !group with initial coordinates of (1000,1000).
ARC1 (0,1), 1000, 0, 1000, -1000, -
                         !Add an arc segment with a center point located at
                         !(1000,0) and the final point located at (1000,-1000)
                         !with negative movement in terms of the encoder.
LINE (0,1), -1000, -1000
                         !Add line segment with final point (-1000,-1000).
ARC2 (0,1), -1000, 0, -3.141529
                         !Add arc segment with center (-1000,0) and a
                         !rotation angle of -π radians.
LINE (0,1), 1000, 1000   !Add line segment with center (1000,1000).
ENDS (0,1)               !Ends the point sequence for the 0 and 1 axes group.
STOP                     !Ends program
```

Figure 11 illustrates the motion created by the example.

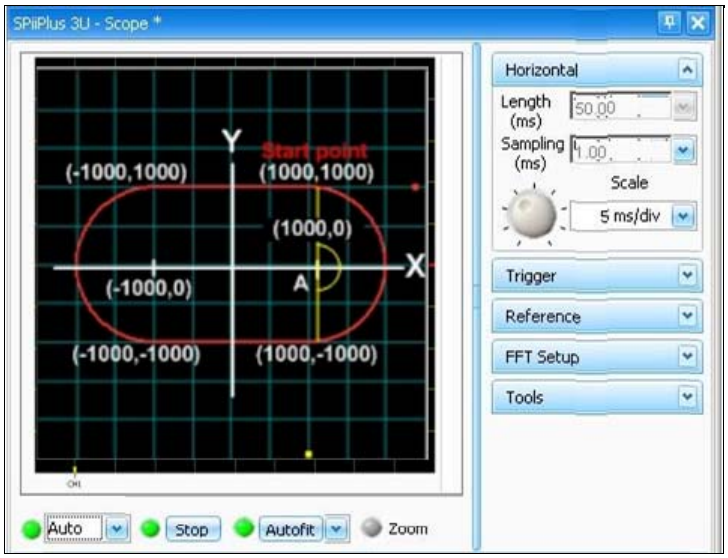


Figure 11 Results of Example MSEG

2.4.9 PATH...ENDS

Description

PATH initiates an arbitrary path motion with linear interpolation using **POINT**, or **MPOINT** for an array of points. The arbitrary path sequence must conclude with **ENDS**.

PATH motion terminates due to:

- Interruption by any new motion command before the current motion concludes terminates the **PATH** motion and causes an error.
- Any fault activation that disables the drive or kills the motion
- User termination by **HALT**, **KILL**, or **DISABLE**

Syntax

PATH*[/command options]axis_list [,time-interval]*

Arguments

axis_list	Axes list, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
time interval	Optional time interval specification in milliseconds. **PATH** specified without **\t** must have the time interval specification as the last argument. The argument defines the time interval between all motion points.

Command Options

/w	Create the motion, but do not start until the **GO** command.
/t	Enables a user-defined time interval in a point array
/c	Use the point sequence as a cyclic array. After arriving at the last point, continue from the first point.
/r	Coordinates of each point are relative to the previous point's coordinates.

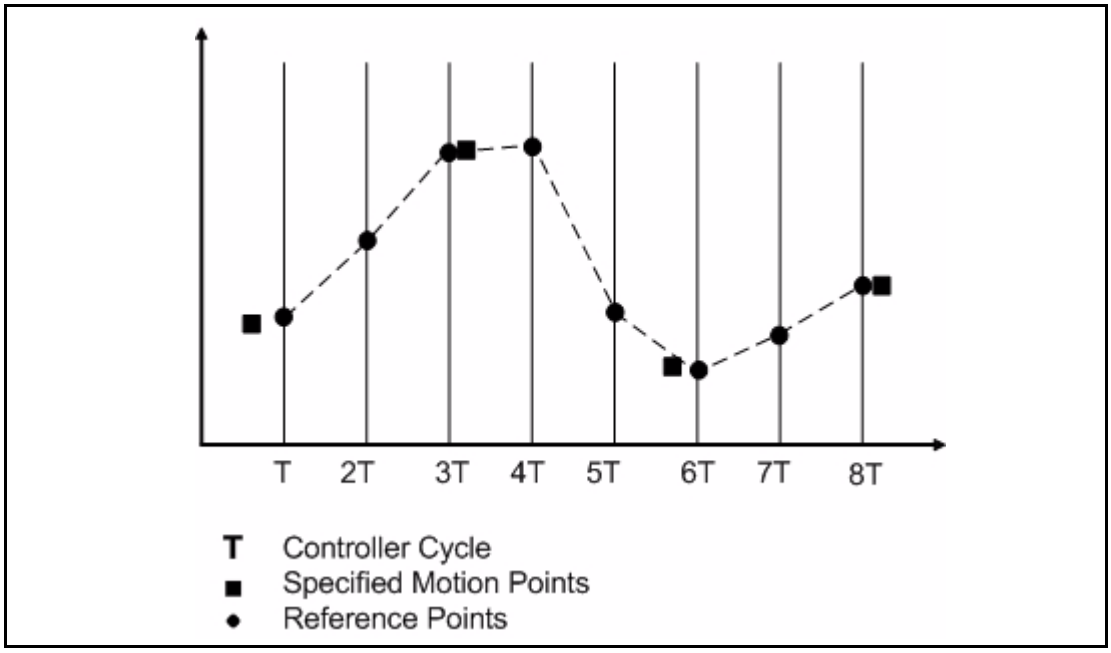


Figure 12 Like PVSPLINE Drawing

Comments
Since a time interval and the destination point are specified, variables **VEL**, **ACC**, **DEC**, **JERK** have no effect on this motion.

Related ACSPL+ Commands
MPTP...ENDS, **POINT**, **MPOINT**, **PVSPLINE...ENDS**

COM Library Methods
ToPoint, ToPointM, ExtToPoint, ExtToPointM

C Library Functions
acsc_ToPoint, acsc_ToPointM, acsc_ExtToPoint, acsc_ExtToPointM

Example
See Example 2 of **MPOINT** for a three axis example of **MPTP...ENDS** motion.

2.4.10 POINT

Description
POINT adds a destination point to multi-point or arbitrary motion paths. A sequence of destination points can be specified with a sequence of **POINT** commands. **POINT** must follow **MPTP...ENDS**, **PATH...ENDS**, or **PVSPLINE...ENDS**. Refer to each command for a list of available command options. The sequence of specified points must conclude with **ENDS**.

Syntax
POINT syntax depends on the command options used in the initializing commands, as follows:

- **POINT** initiated by **MPTP**:

POINT *axis_list, axis_list destination positions*

- **POINT** initiated by **MPTP**/v:

  POINT *axis_list, axis_list destination positions, vector velocity (GVEL)*

  See Example 1.

- **POINT** initialized by **PATH**:

  PATH*[/command options]axis_list, time-interval*

  POINT *axis_list, axis_list destination positions*

  See Example 2.

- **POINT** initialized by **PATH/t**:

  POINT *axis_list, axis_list destination positions, time interval in milliseconds*

  See Example 3.

- **POINT** initialized by **PVSPLINE**:

  PVSPLINE*[command options]axis_list, time-interval*

  POINT *axis_list, axis_list destination positions*

  See Example 4.

- **POINT** initialized by **PVSPLINE/t**:

  PVSPLINE*[command options]axis_list*

  POINT *axis_list, axis_list destination positions, velocity at destination position, time interval in milliseconds*

  See Example 5.

Comments

1. **POINT** axis specifications must follow the same order as the initializing **MPTP**, **PATH** or **PVSPLINE** commands.
2. All arguments following the *axis_list* must appear as a comma separated list, and must correspond to the number of axes. See for an illustration of this rule.

Related ACSPL+ Commands

MPOINT, **MPTP...ENDS**, **PATH...ENDS**, **PVSPLINE...ENDS**

COM Library Methods

MultiPoint, MultiPointM, Spline, SplineM, AddPVPoint, AddPVPointM, AddPVTPoint, AddPVTPointM

C Library Functions

acsc_MultiPoint, acsc_MultiPointM, acsc_Spline, acsc_SplineM, acsc_AddPVPoint, acsc_AddPVPointM, acsc_AddPVTPoint, acsc_AddPVTPointM

Examples

Example 1:

```
MPTP/v (0,1,5)          !Initiates MPTP motion for axes 0, 1 and 5.
                        !simultaneously with a user-defined velocity(GVEL)
                        !at each point.
POINT (0,1,5), 1000,4000,6000,7000
                        !Defines destination points 1000, 4000, and 6000 for
                        !axes 0, 1, and 5 with a GVEL of 7000 units.
ENDS (0,1,5)            !Ends MPTP/v point sequence for axes 0, 1, and 5.
STOP                    !Ends program
```

Example 2:

```
PATH/r (0,1,5),1000     !Initiates PATH motion for axes 0, 1, and 5.
                        !simultaneously where each destination point is
                        !relative to the previous point and the time
                        !interval between each point is one second.
POINT (0,1,5), 1000,2000,3000
                        !Defines destination points 1000, 2000, and 3000 for
                        !axes 0, 1, and 5 respectively.
ENDS (0,1,5)            !Ends PATH point sequence for axes 0, 1, and 5.
STOP                    !Ends program
```

Example 3:

```
PATH/t (0,1,5)          !Initiates PATH motion for axes 0, 1, and 5.
                        !simultaneously with a user-defined time interval
                        !between points.
POINT (0,1,5), 1000,2000,3000, 500
                        !Defines destination points 1000, 2000, 3000 for axes
                        !0, 1, and 5, respectively and a time interval of
                        !500 msec between points.
ENDS (0,1,5)            !Ends PATH point sequence for axes 0, 1, and 5.
STOP                    !Ends program
```

Example 4:

```
PVSPLINE (0,1,5), 10     !Initiates PVSPLINE motion for axes 0, 1, and 5.
                         !Points are given at 10 msec intervals.
POINT (0,1,5), 200, 100, 300, 1000, 2000, 1500
                         !Defines points for axes 0, 1, and 5 with respective
                         !values of 200, 100, 300 and velocities at each
                         !point 1000, 2000, 1500, respectively.
ENDS (0,1,5)             !Ends PVSPLINE point sequence.
STOP                     !Ends program
```

Example 5:

```
PVSPLINE/t (0,1,5)       !Initiates PVSPLINE motion for axes 0, 1, and 5, with
                         !a user-defined time interval between points.
POINT (0,1,5), 200, 100, 300, 1000, 2000, 1500, 250
                         !Defines destination points for axes 0, 1, and 5
                         !with respective values of 200, 100, 300 and
                         !velocities at each respective point of 1000, 2000,
                         !1500. The time interval between points is 250 msecs.
ENDS (0,1,5)             !Ends PVSPLINE point sequence for axes 0, 1, and 5.
STOP                     !Ends program
```

2.4.11 PROJECTION

Description

PROJECTION is an expansion command to the **MSEG...ENDS** set of commands, that allows the controller to perform a three dimensional segmented motion such as creating arcs and lines on a user-defined plane. The method for this 3D segmented motion is to set a transformation matrix that defines a new plane for the segmented motion.

Syntax

PROJECTION *axes_list, Transformation Matrix*

Arguments

axes_list	List of axes, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
Transformation Matrix	Defines the new plane

Comments

1. **MSEG** must precede **PROJECTION**.
2. Prior to using **PROJECTION**, the related axes must be grouped using **GROUP**.
3. The motion parameters of the segmented motion with **PROJECTION** are calculated based on the leading axis described in the **GROUP** command. The other involved axes motion parameters are not relevant. The parameters are calculated to meet a uniform travel time to all grouped axes.

Related ACSPL+ Commands

MSEG...ENDS, **LINE**, **ARC1**, **ARC2**, **GROUP**

COM Library Methods

Projection

C Library Functions

acsc_Projection

Example

1. This example program creates a segmented motion on a new plane (in this example - a circle) on a plane rotated around Axis X by 70° relative to plane AX, as illustrated in **Figure 13**.
2. **ARC2** defines the circle's center coordinates on plane AX. **PROJECTION** transforms these coordinates to the new plane, based on the values in the transformation matrix (Matrix M, in this example).
3. Populate the transformation matrix with the values from table given below. The first two rows define the relationship between the X and Y coordinates of the last **MSEG...ENDS** motion and their respective axes in the new plane. The third row defines the tangent angle of the new plane. In this example uses a 70° angle in reference to Axis A, where tan70=2.74.

Table 17 Matrix Values

Axis	X Coordinate	A Coordinate
X	1	0
A	0	1
B	0	2.74

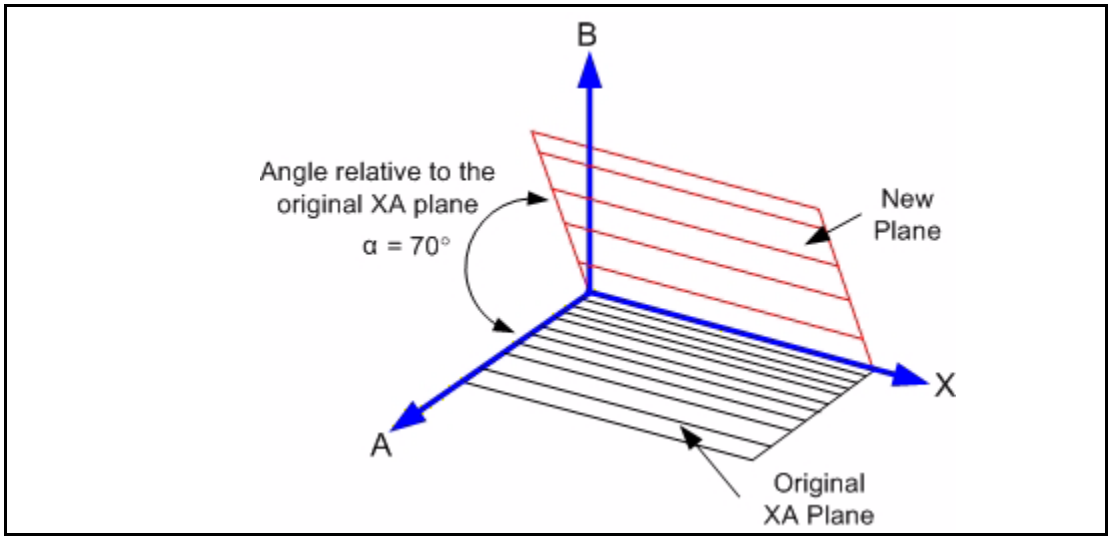


Figure 13 PROJECTION of the XA Plane

Figure 14 illustrates the reference position of axes 0, 4, and 5 (corresponding to the XAB coordinates illustrated in **Figure 13**).

The following screen grab illustrates the reference position of axes 0, 4, and 5.

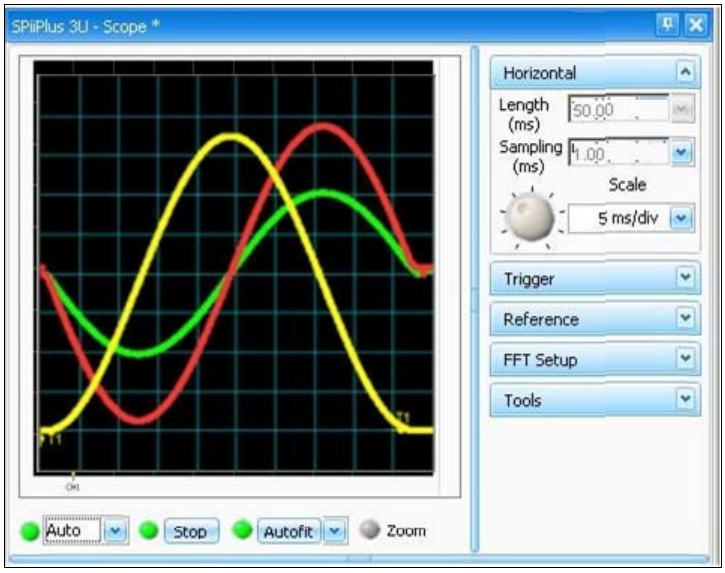


Figure 14 FPOS - PROJECTION Example

Figure 15 illustrates the circle's trajectory viewed from the XA plane.

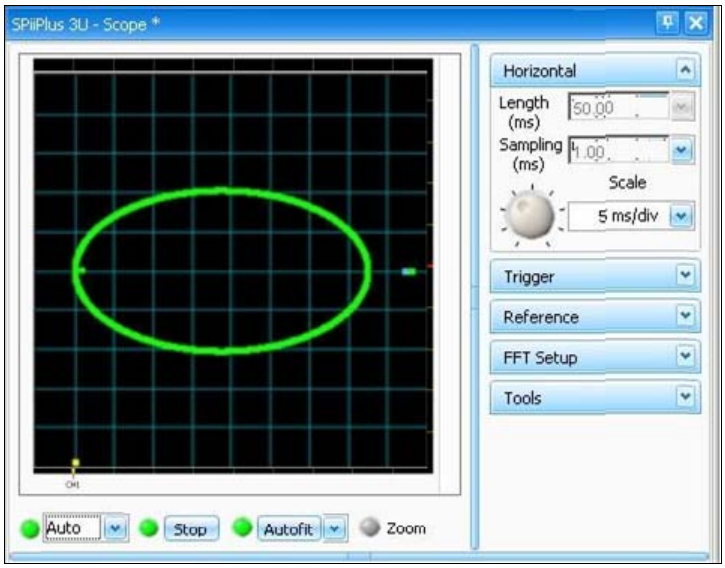


Figure 15 PROJECTION Example - Final Result

```
REAL M(3)(2)              !Defines Transformation Matrix M as real array
! ------------ Set the transformation matrix values----------------------
M(0)(0)=1;M(0)(1)=0;M(1)(0)=0;M(1)(1)=1;M(2)(0)=0;M(2)(1)=2.74
VEL(0)=1000;ACC(0)=10000;DEC(0)=10000
                          !Axis motion parameters
ENABLE (0,4,5)            !Enable the 0, 4 and 5 axes (required)
                          !Corresponding to XAB coordinates shown in Figure 15.
GROUP (0,4,5)             !Group the 0, 4 and 5 axes (required)
SET FPOS(0)=0;SET FPOS(4)=0;SET FPOS(5)=0
                          !Sets the FPOS for 0, 4 and 5, respectively, to 0.
MSEG (0,4),0,0            !Define original plane.
PROJECTION (0,4,5),M      !PROJECTION of axes XAB by matrix M.
ARC2 (0,4),750,0,6,24     !ARC2 performed on new plane.
ENDS (0,4)                !Concludes MSEG.
STOP                      !End Program
```

2.4.12 PTP

Description

PTP (point-to-point) generates motion for the specified axis or axes to a specified destination point.

When **PTP** specifies a single axis, the motion profile is calculated according to **VEL**, **ACC**, **DEC**, **JERK** values of the axis.

In group motion, when **PTP** specifies multiple axes, the group motion profile is based on the leading axis' **VEL**, **ACC**, **DEC**, **JERK** motion values, unless **PTP/m** is used.

PTP terminates due to:

- ❑ Interruption by any new motion command
- ❑ Any fault activation that disables the drive or kills the motion
- ❑ User termination by **HALT**, **KILL**, or **BREAK**.

Syntax

PTP*[command options]axis_list, destination-point[,value for v, value for f]*

Arguments

axis_list	Single axis or axis group, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
destination point	Final destination.
value for v	Optional argument for user-defined velocity.
value for f	Optional argument for user-defined velocity at a destination point

Command Options

/e	Wait for motion termination before executing next command.
/f	Specify non-zero velocity at each destination point (or points) in a series of PTP motions
/m	Use the motion profile values of the axis group as a whole, rather than those of the leading axis, without exceeding any of the defined axes motion **VEL**, **ACC**, **DEC**, **JERK** values.
/r	The destination point is relative to the start point.
/v	Use the specified velocity instead of the default velocity (**VEL**).
/w	Create the motion, but do not start until GO.

Comments

1. Axes destination points, and relative velocity in the **PTP** command can also be an expression.
2. Multi axes **PTP** executes **PTP** motion for a group of axes. For example, *PTP XYZ* creates motion for axes X, Y, and Z as a group.

Related ACSPL+ Commands

MPTP...ENDS, **POINT**

COM Library Methods
ToPoint

C Library Functions
acsc_ToPoint

Examples
Example 1:

```
PTP/v 1, 2000, 500      !PTP axis 1 to point 2000 with velocity 500
```

Example 2:

```
PTP/rw (0,1), 1000,     !PTP axes 0 and 1 where the 1 target point is 1000
2000
                        !and the 0 target point is 2000. The target points
                        !are relative to the start point. Motion will not
                        !commence until GO command is issued.
```

Example 3:

```
ENABLE 0                !Enables axis 0
VEL(0)=10000            !Sets the default axis 0 velocity to 10000
SET RPOS(0)=0           !Sets the current axis 0 position to 0
PTP/rf 0, 2000, 1000    !Initiates a relative axis 0 motion of 2000 with end
                        !velocity of 1000 at the destination point
PTP 0, 4000             !Initiates an absolute axis 0 motion to 4000
STOP                    !Ends program
```

2.4.13 PVSPLINE...ENDS

Description
PVSPLINE (position-velocity spline) creates an arbitrary motion trajectory where the controller provides cubic spline interpolation between two points. The user specifies the end point and the end velocity for each motion segment. **ENDS** must terminate the point sequence.

PVSPLINE motion terminates due to:

- Interruption by any new motion command
- Any fault activation that disables the drive or kills the motion
- User termination by **HALT**, **KILL**, or **DISABLE**

Syntax
PVSPLINE*[command options]axis_list[,time-interval]*

Arguments

axis_list	Single axis or axis group, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
time-interval	Optional argument for user-defined time interval, in milliseconds, between points.

Command Options

/r	The point coordinates are relative to the previous point
/w	Create the motion, but wait to start for **GO**
/t	Non-uniform time interval: time interval is specified for each point along with the point coordinates
/c	Perform the point sequence as a cyclic array-after the last point, start the motion again from the first point.

Comments

1. Since the time interval and the destination point are defined, variables **VEL**, **ACC**, **DEC**, and **JERK** have no effect on **PVSPLINE**.
2. If **PVSPLINE** motion is ended with **HALT**, the controller does not follow the motion trajectory during deceleration.
3. **PVSPLINE** specified with /**t** must **NOT** have a *time-interval* specification. Instead, specify the time interval for each point as an additional argument for **POINT** or as an additional array row in **MPOINT**.

Related ACSPL+ Commands

POINT, **MPOINT**

COM Library Methods

Spline, SplineM, AddPVPoint, AddPVPointM, AddPVTPoint, AddPVTPointM

C Library Functions

acsc_Spline, acsc_SplineM, acsc_AddPVPoint, acsc_AddPVPointM, acsc_AddPVTPoint, acsc_AddPVTPointM

Example

```
REAL ARRAY_NAME(4)(5)    !Defines a point array of four rows and five columns
                         !and a variable that defines the number of array rows
                         !that are part of the motion.
INT NUMBER_of_POINTS     !Defines NUMBER_of _POINTS as an integer variable.
NUMBER_of_POINTS=3       !Assigns 3 to the NUMBER_of_POINTS variable.
PVSPLINE/r (0,1),10      !Initiates PVSPLINE motion for axis 0 and 1 with
                         !relative destination points and a time interval of
                         !10 milliseconds between each point.
POINT (0,1),200,100,1000,2000
```

```
                        !Defines values for 0 and 1 axes where:
                        !200, 100=destination point for the 0 and 1 axes,
                        !respectively; and 1000, 2000=velocity at specified
                        !points for the 0 and 1 axes, respectively.
MPOINT (0,1), ARRAY_NAME, NUMBER_of_POINTS
                        !Defines an MPOINT array for the 0 and 1 axes, where
                        !ARRAY_NAME is called, and where NUMBER_of_POINTS
                        !defines the number of array columns involved in the
                        !motion.
ENDS (0,1)              !Ends MPOINT for 0 and 1 axes.
STOP                    !Ends program.
```

Figure 16 illustrates a typical **PVSPLINE** motion.

The following diagram illustrates a typical **PVSPLINE** motion.

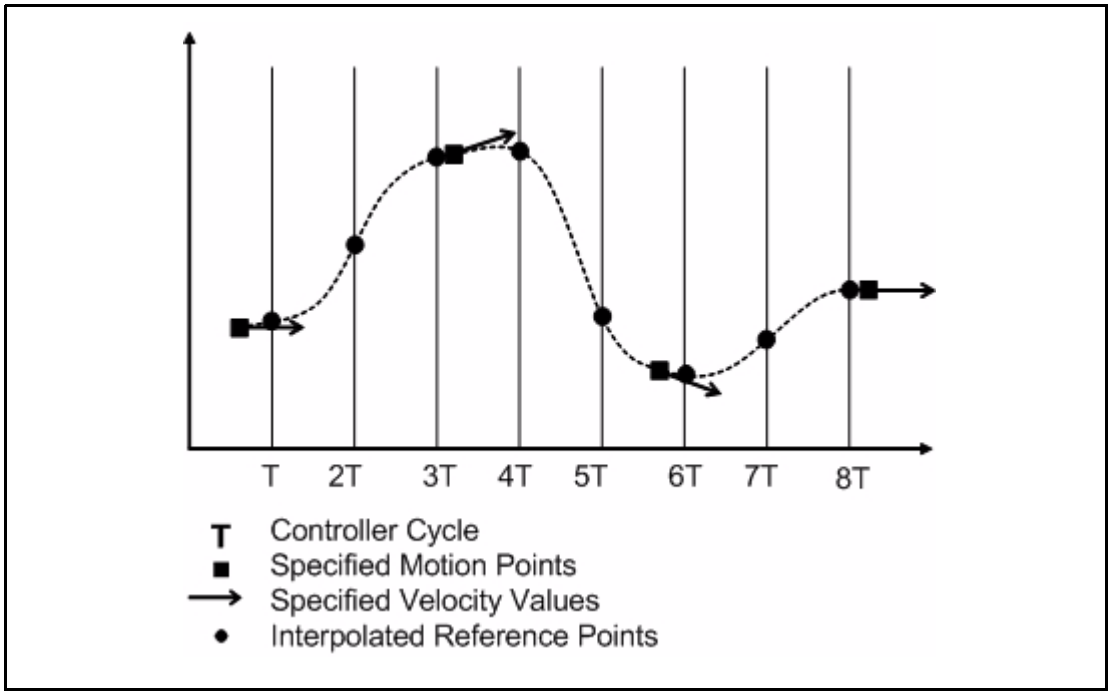


Figure 16 PVSPLINE Motion

2.4.14 SLAVE

Description

SLAVE initiates a motion based on position lock or velocity lock slaved to a master value or expression. Only individual axes are allowed. The initiated motion starts immediately if the axis is idle, otherwise the motion waits in the motion queue until all previously created motions finish. **SLAVE** must precede **MASTER**.

MASTER - SLAVE dependency terminates with:

- ❑ Any fault activation that disables the drive or kills the motion
- ❑ **HALT**, **KILL**, **BREAK**
- ❑ **DISABLE** to the **SLAVE** axis.
- ❑ Setting the logical dependence between master and slave axes to zero. For example MASTER MPOS(0)=0

Syntax

SLAVE*[/command options] axis[,lower boundary, upper boundary]*

Arguments

axis	Axis designation, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
lower boundary	Optional argument used for setting the lower boundary of master-dependent SLAVE motion.
upper boundary	Optional argument used for setting the upper boundary of master-dependent SLAVE motion.

Command Options

/w	Create the motion, but do not start until **GO** is executed.
/p	Master - Slave motion generated by position lock.
/t	Create slave motion within specified position boundaries.

Comments

1. When **/p** is used, the controller first initiates **velocity lock**. Only after the achieving **velocity lock** the controller will engage **position lock**.
2. If no command option is specified, the default mode is **velocity lock**.

Related ACSPL+ Commands

MASTER, **GO**, **HALT**, **KILL**, **DISABLE**

Related ACSPL+ Variables

XSACC, **SYNV**, **MFF**, **JERK**, **ACC**, **VEL APOS**

COM Library Methods

SetMaster, Slave, SlaveStalled

C Library Functions
acsc_SetMaster, acsc_Slave, acsc_SlaveStalled

Example
See **MASTER** for an example of **MASTER - SLAVE** syntax.

2.4.15 STOPPER

Description
STOPPER is used in conjunction with **MSEG...ENDS** to avoid velocity jumps at segment inflection points. When **STOPPER** is specified between two segments, the controller provides smooth deceleration to zero before **STOPPER** and a smooth acceleration to the default or specified velocity after **STOPPER**.

Syntax
STOPPER *axis_list*

Arguments

axis_list	Single axis or axis group, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.

Related ACSPL+ Commands
MSEG...ENDS, **ARC1**, **ARC2**, **LINE**, **PROJECTION**

COM Library Methods
Stopper

C Library Functions
acsc_Stopper

Example
Figure 17 illustrates the following example. For this illustration the reference position (**RPOS**) for axes 0 and 1 was set to (0,0).

```
MSEG (0,1), 0, 0          !MSEG initiates segmented motion for the 0 and 1 axis
                          !group with the point on the plane located at (0, 0).
LINE (0,1), 1000, 2500    !Add line segment with final point at (1000, 2500).
STOPPER (0,1)             !Slow down to zero.
ARC1 (0,1), 0,2000, -1000,2500, +
                          !Add arc segment with final point (-1000, 2500) and
                          !center point (0, 2000).
STOPPER (0,1)             !Slow down to zero.
LINE (0,1), 0, 0          !Add line segment with final point (0, 0).
ENDS (0,1)                !End MSEG.
LINE (0,1), 1000, 1000    !Add line segment with final point (1000, 1000).
ENDS (0,1)                !Ends the point sequence for the the 0 and 1 axis
                          !group.
```

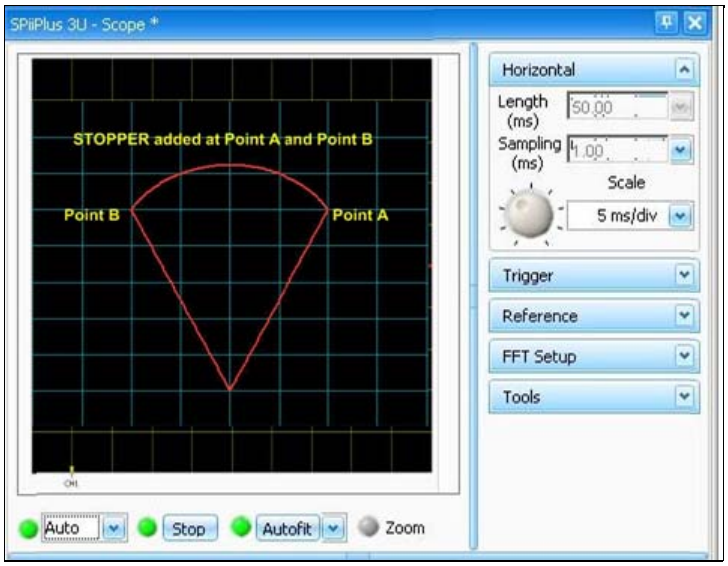


Figure 17 Use of STOPPER

2.4.16 TRACK

TRACK initiates track motion. In **TRACK** motion, a new point-to-point move is generated to a new target position whenever the variable **TPOS** (target position) changes. **TRACK** does not terminate automatically. If **TPOS** is not assigned a new value, motion stops at the last **TPOS** and waits. If a new **TPOS** value is assigned, motion continues.

TRACK terminates due to:

- ❑ Any subsequent motion command except **TRACK**.
- ❑ Any fault activation that disables the drive or kills the motion.
- ❑ User termination by **HALT**, **KILL**, or **DISABLE**.

Syntax

TRACK *[command options] axis*

Arguments

axis	Axis designation, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.

Command Options

/w	Create the motion, but do not start until **GO**.

- Comments

1. While **PTP** code appears shorter and simpler, there are applications where **TRACK** is preferable to **PTP**. For example, **TRACK** provides an easy way to change the destination

position at any time during the motion by changing the target position (**TPOS**) variable. The controller terminates the current motion and proceeds to the next destination point, on-the-fly.

2. **TRACK** is for single axis motion only.
3. **TPOS** is updated every controller cycle.

Related ACSPL+ Commands
PTP

COM Library Methods
Track

C Library Functions
acsc_Track

Example

```
TRACK 0                 !Initiates TRACK for 0 axis.
TPOS(0) = NewTarget     !The controller generates a PTP motion to the point
                        !designated by NewTarget.
```

2.5 Program Flow Commands

The Program Flow commands are:

Command	Action
Assignment Command	Assigns values
BLOCK...END	Executes a group of commands in one MPU cycle
CALL	Calls subroutine.
GOTO	Transfers program execution to another point in the program.
IF, ELSEIF, ELSE...END	**IF** command structure.
INPUT	Suspends program execution pending user input
LOOP...END	Loop command structure.
ON...RET	Defines an autoroutine
TILL	Delays program execution until a specified expression produces a non-zero (true) result.
WAIT	Delays program execution for a specified number of milliseconds.
WHILE...END	While command structure.

2.5.1 Assignment Command

Description

The **Assignment** command (=) is used to assign a value to ACSPL+ standard or user-defined variables with read/write options.

Syntax

variable_name = *value*

Arguments

variable_name	Can be: ❑ Name of ACSPL+ standard or user-defined variable ❑ An element of an ACSPL+ or user array ❑ One bit of integer variable or integer array element (applicable only to those variables having specifically named bits, for example, **IMASK**)
value	Can be of integer or real type. The ***value*** argument can either be: ❑ A specific value ❑ An expression that during runtime calculates to a value

Comments

Assigning to an ACSPL+ variable is limited by the following rules:

- ❑ Assignment to read-only variable (for example, **FPOS**) is prohibited
- ❑ Assignment to a protected variable (for example, **ERRI**) is allowed in only in the Configuration mode.

After assignment, the previous value of the variable is replaced by the new value.

User local and global variables must be declared before they can be used in an assignment command.

- ❑ Explicit indexing only is allowed for user array variables.
- ❑ If a user variable is scalar, no indexing is required.
- ❑ If a user variable is one-dimensional array, it requires one index. Two-dimensional arrays require two indexes.

A bit can have only two possible values: 0 (false) or 1 (true), while the ***value*** result, which defines the bit value, can be any value. Assignments convert the value as follows:

- ❑ If the value is zero, the bit is set to zero
- ❑ If the value is non-zero, the bit is set to one

Although bit assignments are applicable to any integer variable or array element, they are mainly used for changing flag variables and output bits.

The controller executes assignment commands in the following order:

1. Calculate ***value***
2. Convert the type of calculated value to the type of ***variable_name*** (if the types differ)
3. Assign the result to ***variable_name***

Examples

```
VEL(0) = 1000           !Assign 1000 to axis 0 default velocity -
                        !explicit indexing.
VEL0 = 1000             !Assign 1000 to axis 0 default velocity -
                        !postfix indexing.
Var1 = FPOS(0)          !Assign value of ACSPL+ variable to user variable.
Var2(0)(5) = 200        !Assign to element of user array.
OUT0.5 = 1              !Assign to digital output 5
```

2.5.2 BLOCK...END

Description

Commands specified within the **BLOCK...END** structure are executed in one MPU cycle.

Syntax

BLOCK

command-list

END

Arguments

command-list	List of commands, separated by semi-colons.

Comments

1. The structure provides an alternative to specifying *command-list* commands in one line. The commands within the structure can be specified in several lines. However, the controller executes all commands in one controller cycle, as if they were written in one line.
2. Commands and functions that may cause delay (**WAIT**, **TILL**, **GETSP**, **WHILE...END**, **LOOP...END** etc.) provide delay even if they are used within the **BLOCK...END** structure.

Example

```
INT XX,YY,ZZ            !Defines variables as integers
BLOCK                   !BLOCK command
      XX=TIME           !XX is assigned the TIME standard variable
      YY=TIME           !(controller timer from power-up, in mSec.
                        !One execution line later, YY is also
                        !assigned the TIME standard variable.
                        !Because both lines were written in the
                        !BLOCK…END block, they are both executed
                        !during the same controller cycle hence the
                        !difference between the two values =0
                        !(ZZ=0).
                        !If both lines were NOT written within the
                        !BLOCK structure, the difference would be
                        !one controller cycle (ZZ=1).
```

```
END                     !End the BLOCK command
ZZ=YY-XX                !The result in this example is ZZ=0.
```

2.5.3 CALL

Description

CALL calls a subroutine according to a specified label. All subroutines must begin with a label and conclude with **RET**.

Syntax

CALL *label*

…

STOP

label:

…

RET

Arguments

label	Unique name identifier.

Related ACSPL+ Commands

GOTO

Comments

1. A label specified by **CALL** must be defined somewhere in the same program buffer. The subroutine starts after the label and spans all commands up to **RET**.
2. **CALL** transfers program execution to the start of the subroutine and stores the return point in the stack. When **RET** executes, the return point is extracted from the stack and execution continues from the next command after **CALL**.
3. Subroutines can be nested; a subroutine can call another subroutine, and so on.
4. **CALL** can only call a sub-routine located in the same buffer

Example

```
! ------------ Start subroutine labeled CHECK_PE ---------------------
CALL CHECK_PE           !Call a subroutine named CHECK_PE.
DISABLE 0               !This line will be executed when the
                        !subroutine is terminated by the RET command.
STOP                    !Ends program.
CHECK_PE:               !Subroutine label name.
WHILE (ABS(PE(0)>20))   !Subroutine command lines.
DISP "PE is :", PE(0)
END                     !End WHILE loop.
RET                     !Terminate subroutine and return to main program.
```

2.5.4 GOTO

Description

GOTO transfers program execution to a particular point in the program specified by a unique label.

Caution	
	Avoid using **GOTO** to enter or exit a subroutine. Failure to do so will result in program termination due to a stack violation error. See **CALL**.

Syntax

GOTO *label*

Arguments

label	A unique label defined in the program buffer.

Comments

1. A label specified by **GOTO** must be defined somewhere in the same program buffer.
2. The next executed command is located after the label.

Related ACSPL+ Commands

CALL

Example

```
GOTO ARR                !Go to a label named ARR
```

2.5.5 IF, ELSEIF, ELSE...END

Description

IF, **ELSEIF**, **ELSE**, and **END** are building blocks used in the **IF** control structures. These commands specify a condition which must be met before executing a list of commands. **IF** control structures must conclude with **END**.

If the condition result is none-zero (true), the command list in the corresponding clause executes and all subsequent conditions are not validated and all other command lists are skipped. After executing the command list, the program continues from the next line after **END**.

If a condition result is zero (false), the command list is skipped. The program then checks subsequent **ELSEIF** conditions. If no condition result is true, the command list following **ELSE** executes, if it exists. The program then continues from the command following **END**.

Table 18 describes the **IF** control structure syntax using these building blocks.

Syntax

Table 18 IF Control Structures

Syntax	IF Structure
IF *expression* command list **END**	IF-END
IF *expression* command list **ELSE** command list **END**	IF-ELSE-END
IF *expression* command list **ELSEIF** sequence END	IF-ELSEIF-END
IF *expression* command list **ELSEIF** sequence **ELSE** *command list* **END**	IF-ELSEIF-ELSE-END

Comments

1. The **IF** -**ELSE** -**END** control structure has two command lists. One follows **IF** and the second follows **ELSE**. Only one command list executes, depending on the result of the expression following **IF**.
2. The **IF**-**ELSEIF**-**END**, and **IF**-**ELSEIF**- **ELSE** -**END** forms provide validation of a sequence of conditions.
3. **IF** control structures may contain any number of **ELSEIF** clauses. Each **ELSEIF** clause specifies its own condition. The conditions are validated in the following order:

   a) Condition after **IF**

   b) Condition after first **ELSEIF**

   c) Condition after second **ELSEIF**

Examples

Example 1:

This example program fragment activates either output 5 or 6 and shuts off the other, depending on the state of input 3.

```
IF IN0.3                        !Check if the third bit of IN0 is 1.
        OUT0.5 = 1              !Set the 5th bit of OUT0 to 1.
        OUT0.6 = 0              !Set the 6th bit of OUT0 to 0.
ELSE                            !Execute the next commands only if the IF
                                !condition was false
        OUT0.5 = 0              !Set the 5th bit of OUT0 to 0.
        OUT0.6 = 1              !Set the 6th bit of OUT0 to 1.
```

```
END                                 !End of the IF body.
STOP                                !Ends program
```

Example 2:

The following program fragment implements a saturation effect by limiting variable **V0** to a range from -256 to +256.

```
IF V0 < -256                        !Check if V0 is less than -256
        V0 = -256                   !Assign -256 to V0 (low saturation)
ELSEIF V0 > 256                     !If V0 is not less than -256, check if it
                                    !is greater than 256
        V0 = 256                    !Assign 256 to V0 (high saturation)
END                                 !End of the IF body (no saturation if
                                    !V0 is equal to or more than 256)
STOP                                !Ends program
```

2.5.6 INPUT

Description

INPUT suspends program execution in a specific buffer until the host or the user (via the **Terminal** option) provides a specific value or values.

Syntax

INPUT*(array)*

Arguments

array	User defined array.

Comments

The INPUT command accepts as an argument a scalar or array variable. The command delays program execution until the host provides an input. The input from the host is expected to consist of one or more numbers separated by spaces or commas and followed by a carriage return. The INPUT command scans the numbers in the input and stores them in the argument variable.

If the argument variable is scalar, INPUT takes the first number in the input and stores it in the argument variable.

If the argument is an array, INPUT fills argument the array with the values from the input.

The function execution is completed when the scalar is stored or the array is filled or a carriage return is encountered.

COM Library Command

SuspendBuffer

C Library Command

acsc_SuspendBuffer

Related ACSPL+ Commands
None

Examples
Example 1:

```
INT ARR(10)             !Declares a ten-element array.
…                       !Other program commands.
INPUT (ARR)             !Wait for user to enter values in ARR array.
```

Program execution halts until input is received from the host.

Assume that a user enters: 20 30 40 <ENTER>, the program stores the three values in ARR(0), ARR(1), and ARR(2) and resumes execution. Since the user has entered only three numbers, the program will store 0 (zero) as the value in the elements: ARR(3) through ARR(9).

Note	*The user may enter the value of a keyboard function key, 0 (zero) followed by an integer from 1 to 12, in which case the value is stored separately in the standard variable: **FK**.*

Example 2:

```
INT ARR(2)              !Declares a two-element array.
DISP "[FILL AXIS NUMBER],[FILL DISTANCE TO GO]"
                        !Display the text in the Terminal
INPUT (ARR)             !This command stops the program until the two
                        !variables are entered into the ARR array.
SET FPOS(ARR(0))=0      !Set the feedback position to zero.
ENABLE (ARR(0))         !Enable the drive specified by the value of ARR(0).
PTP/RE (ARR(0)), (ARR(1))
                        !Move the ARR(0) axis to position  ARR(1)
```

2.5.7 LOOP...END

Description
The **LOOP** command structure provides a fixed number of command list repetitions set by the definition of an exact value, or any expression. **LOOP** control structures must conclude with **END**.

Syntax
LOOP *expression*

command_list

END

Arguments

expression	Defines the number of loops. It can also be a specific integer number.
command_list	Any set of commands that are to be repeated.

Comments

1. If the ***expression*** result is zero or negative, the command list is not executed and the program continues from the next line after **END**.
2. If the ***expression*** result is not an integer the command rounds-off the number to the closest integer.

Examples

Example 1:

```
LOOP 10                 !Do 10 repetitions.
J=J+1                   !J is incremented by 1 for each repetition.
END                     !End LOOP.
STOP                    !Ends program.
```

Example 2:

```
LOOP ARR                !Do repetitions equal to value labeled by ARR.
J=J+1                   !J is incremented by 1 for each repetition.
END                     !End LOOP.
STOP                    !Ends program.
```

Example 3:

```
LOOP YY*325/TT          !Do repetitions equal to the value of the
                        !expression YY*325/TT
J=J+1                   !J is incremented by 1 for each repetition.
END                     !End LOOP.
STOP                    !Ends program.
```

2.5.8 ON...RET

Description

ON...RET defines an autoroutine. An autoroutine consists of a condition, and a body. Autoroutines must conclude with **RET**. The autoroutine condition is checked every MPU cycle. Once the autoroutine condition is met, the autoroutine interrupts, executes the lines in the body (until **RET)**, and then transfers execution control back to the interrupted program line.

Caution	
	*The controller should never directly execute **ON**. If the program execution flow comes to **ON**, the controller asserts a runtime error and aborts the program. To avoid this error, use **ON** after **STOP**.*

Syntax

ON *condition*

auto_routine body

RET

Comments

1. Once the buffer in which an autoroutine is compiled, that autoroutine is enabled.
2. The controller implements edge-detection in autoroutine condition verification. If a condition becomes true, the controller activates the autoroutine only once. If afterwards the condition remains true, the controller does not activate the autoroutine again. The condition must become false and then become true again in order to activate the autoroutine again.
3. Only one autoroutine can be active in a buffer at a time.

Related ACSPL+ Commands

ENABLEON, **DISABLEON**

Examples

Example 1:

Demonstrates a typical use of an autoroutine for processing controller faults. The autoroutine provides an error message when a drive alarm on X axis occurs.

```
STOP                                !STOP must precede the autoroutine.
ON FAULT(0).#DRIVE                  !Activate autoroutine when bit
                                    !FAULT(0).#DRIVE changes from 0 to 1
DISABLE 0                           !Disable drive X
DISP "0 Axis Drive Alarm"           !Display message
RET                                 !End of autoroutine
```

Example 2:

Illustrates how all variables, not only faults, can be used in autoroutine conditions. Assuming that output OUT0.6 is connected to a LED indicator, the following autoroutine signals the motion state bit to activate the indicator, and deactivate it when the axis is no longer in motion.

```
STOP                          !STOP must precede the autoroutine.
ON MST(0).#MOVE               !When MST(0).#MOVE bit changes from 0 to
                              !1 (signaling that the axis is moving)
OUT0.6 = 1                    !Set output 6 to 1 (turn on the LED)
RET                           !End of autoroutine
ON ^MST(0).#MOVE              !When MST(0).#MOVE bit changes from 1 to
                              !0 (signaling that the axis is no longer
                              !moving)
OUT0.6 = 0                    !Set output 6 to 0 (turn off the LED)
RET                           !End of autoroutine
```

2.5.9 TILL

Description

TILL delays program execution until the result of a specified expression is met, or the return value is non-zero.

Syntax

TILL *expression [,time_out]*

Arguments

expression	Defines how long to pause the program execution
time_out	Used to specify a time-out in milliseconds. Use ***time_out*** to limit the time that an expression can return zero.

Related ACSPL+ Commands

WAIT

Examples

Example 1

```
TILL ^MST(0).#MOVE      !Wait for 0 axis termination
```

Example 2

```
TILL IN0.0=1, 2000      !Wait until input #0 =1 or 2000 milliseconds
```

Example 3

```
TILL RPOS(0)<>0         !Wait until RPOS(0) is not equal to 0
```

2.5.10 WAIT

Description

WAIT delays program execution for the specified number of milliseconds.

Syntax

WAIT *wait_time*

Arguments

wait_time	Defines how long to delay the program execution. ***wait_time*** can also be defined by an expression.

Related ACSPL+ Commands

TILL

Examples

Example 1

```
WAIT 2000              !Wait 2000 milliseconds
```

Example 2

```
WAIT YY=65/TT          !Milliseconds defined by an expression
```

2.5.11 WHILE...END

Description

The **WHILE** command structure provides repetitive execution of commands list as long as a condition is satisfied. If the condition was not satisfied when checked for the first time, program execution continues from the command following **END**.

WHILE control structures must conclude with **END**.

Syntax

WHILE *condition*

command_list

END

Arguments

condition	Condition controlling the execution of the WHILE ***command_list***.
command_list	List of commands to be executed.

Related ACSPL+ Commands

IF, ELSEIF, ELSE...END, **TILL**

Examples
Example 1:

```
WHILE S_ST.#DC                    !Indicate data collection active.
OUT0.0=^OUT0.0                    !Blink LED.
WAIT 200                          !Blink period is 200msec.
END
```

Example 2:

```
WHILE (ABS(PE(0)>20))             !Do if the 0 axis Position Error is greater
                                  !than 20.
DISP "PE is :", PE(0)             Display the 0 axis Position Error
END
```

Example 3:

```
WHILE 1                           !Run forever
PTP/e 0, 1000
PTP/e 0, -1000
END
```

2.6 Program Management Commands

The Program Management commands are:

Command	Action
DISABLEON	Disables autoroutine activation in a buffer.
ENABLEON	Enables autoroutine activation in a buffer.
PAUSE	Suspends program execution in a buffer.
RESUME	Resumes program execution in a buffer.
START	Activates program execution in a buffer.
STOP	Terminates program execution in a buffer. STOPALL terminates all programs.

2.6.1 DISABLEON

Description
DISABLEON disables autoroutine activation in a buffer. The command has the same functionality as **PFLAGS**.**#NOAUTO**=1.

Syntax
DISABLEON *[(buffer-number)]*

Arguments

buffer-number	Optional argument, if no buffer number is mentioned, the command disables autoroutines in all buffers.

Related ACSPL+ Commands
ON...RET, **ENABLEON**

2.6.2 ENABLEON

Description
ENABLEON enables autoroutine activation in a buffer. The command has the same functionality as **PFLAGS**.**#NOAUTO**=0.

Syntax
ENABLEON *[(buffer-number)]*

Arguments

buffer-number	A number between 0 and 15 specifying the buffer in which the autoroutine is to be enabled.. If no buffer number is given, the command enables autoroutines in all buffers.

Related ACSPL+ Commands
ON...RET, **ENABLEON**

2.6.3 PAUSE

Description
PAUSE suspends program execution in a specific buffer.

Syntax
PAUSE *buffer-number*

Arguments

buffer-number	A number between 0 and 15 specifying the buffer in which the program is to be suspended.

Related ACSPL+ Commands
RESUME

Example

```
PAUSE 5                    !Pauses buffer five
```

2.6.4 RESUME

Description
RESUME resumes the program execution in a specific buffer after the execution was paused.

Syntax
RESUME *buffer-number*

Arguments

buffer-number	A number between 0 and 15 specifying the buffer in which the program execution that was suspended by the **PAUSE** command is to resumed.

Related ACSPL+ Commands
PAUSE

Example

```
RESUME 5                   !Resume buffer five
```

2.6.5 START

Description
START activates program execution at a specified line number, or a specific label in a specified buffer that is different from the buffer where **START** is enabled.

Syntax
START *buffer-number, line-number | label*

Arguments

buffer-number	A number between 0 and 15 specifying the buffer.
line-number | label	The line number of the program within the buffer where execution is to begin, or a label identifying the line number.

Comments
1. Unless line #1 is used in the **START** command, it is recommended to use a label since line numbers are prone to change if programs lines are deleted or inserted.
2. **START** executes successfully if the target buffer is loaded with a program, compiled, but not running. Otherwise, **START** causes a run-time error and aborts the current program.
3. The program activated by **START** executes concurrently with the program containing the **START** command, and other active programs.

Related ACSPL+ Commands
STOP, **BREAK**

Example
Example 1

```
START 8,15              !Start program in buffer 8 line 15
```

Example 2

```
START 8,Start_here      !Start program in buffer 8, at the line identified
                        !by the Start_here label.
```

2.6.6 STOP

Description
STOP terminates program execution in the specified buffer. **STOPALL** terminates program execution in all buffers.

Syntax
STOP *buffer-number*

Arguments

buffer-number	A number between 0 and 15 specifying the buffer in which the program execution is to be halted.

COM Library Command
StopBuffer

C Library Command
acsc_StopBuffer

Example

```
STOP 5                  !Terminate program execution in buffer five.
```

3 ACSPL+ Variables

This chapter covers the ASCPL+ variables. ACSPL+ has a complete set of built-in variables for use in setting values that control ACSPL+ programs. The ACSPL+ variables are divided into the following categories:

- **Axis Configuration Variables**
- **Axis State Variables**
- **Data Collection Variables**
- **Input and Output Variables**
- **Monitoring Variables**
- **Motion Variables**
- **Program Execution Control Variables**
- **Safety Control Variables**
- **Induction Motor Variables**
- **Nanomotion Variables**
- **Servo-Loop Variables**

  Along with the following groups:
  - **Servo-Loop Current Variables**
  - **Servo-Loop Velocity Variables**
  - **Servo-Loop Velocity Notch Filter Variables**
  - **Servo-Loop Velocity Low Pass Filter Variables**
  - **Servo-Loop Velocity Bi-Quad Filter Variables**
  - **Servo-Loop Position Variables**
  - **Servo-Loop Commutation Variables**
  - **Servo-Loop Compensations Variables**
  - **Servo-Loop Miscellaneous Variables**
- **System Configuration Variables**
- **Communication Variables**
- **Miscellaneous**

The ACSPL+ programming variables are:

Table 19 Alphabetical Listing of All ACSPL+ Variables (page 1 of 6)

Name	Description	Variable Group
ACC	Acceleration	Motion
AERR	Axis Error	Safety Control
AFLAGS	Axis Flags	Axis Configuration

Table 19 Alphabetical Listing of All ACSPL+ Variables (page 2 of 6)

Name	Description	Variable Group
AIN	Analog Inputs	Input and Output
AOUT	Analog Outputs	Input and Output
APOS	Axis Position	Axis State
AST	Axis State	Axis State
BAUD	Serial Communication Baud Rate	Communication
BOFFTIME	Brake Deactivation Time	Axis Configuration
BONTIME	Brake Activation Time	Axis Configuration
CERRA	Critical Position Error in Accelerating	Axis Configuration
CERRI	Critical Position Error in Idle	Axis Configuration
CERRV	Critical Position Error in Moving	Axis Configuration
CFG	Configuration Mode	System Configuration
COMMCH	Communication Channel	Communication
COMMFL	Communication Flags	Communication
CONID	Controller Identification	Communication
CTIME	Controller Cycle Time	System Configuration
DAPOS	Delayed Axis Position	Axis State
DCN	Axis DC, Number of Samples	Data Collection
DCOM	Drive Command	Servo-Loop
DCP	Axis DC, Period	Data Collection
DEC	Deceleration	Motion
DELI	Delay on Transition to Idle State	Axis Configuration
DELV	Delay on Transition to Velocity State	Axis Configuration
DISPCH	Default Communication Channel	Communication
E_FREQ	Primary Encoder Frequency	Axis Configuration
E_SCMUL	Primary Encoder Sin-Cos Multiplier	Axis Configuration
E_TYPE	Primary Encoder Type	Axis Configuration
E2FAC	Secondary Encoder Factor	Axis Configuration
E2OFFS	Secondary Encoder Offset	Axis Configuration
ECERR	Contains EtherCAT Error Code	Communication
ECHO	Echo Communication Channel	Communication
ECST	Contains EtherCAT Status	Communication
EFAC	Primary Encoder Factor	Axis Configuration
ENTIME	Enable Time	Axis Configuration
EOFFS	Primary Encoder Offset	Axis Configuration
ERRA	Position Error in Accelerating	Axis Configuration
ERRI	Position Error in Idle	Axis Configuration
ERRV	Position Error in Moving	Axis Configuration
EXTIN	Extended Inputs (HSSI)	Input and Output

Table 19 Alphabetical Listing of All ACSPL+ Variables (page 3 of 6)

Name	Description	Variable Group
EXTOUT	Extended Outputs (HSSI)	Input and Output
F2POS	Secondary Feedback Position	Axis State
F2VEL	Secondary Feedback Velocity	Axis State
FACC	Primary Feedback Acceleration	Axis State
FAULT	Faults	Safety Control
FDEF	Default Response Mask	Safety Control
FK	Function Key	Miscellaneous
FMASK	Fault Mask	Safety Control
FPOS	Primary Feedback Position	Axis State
FVEL	Primary Feedback Velocity	Axis State
FVFIL	Primary Feedback Velocity Filter	Axis Configuration
GACC	Group Acceleration	Motion
GJERK	Group Jerk	Motion
GMOT	Motion Number	Motion
GMQU	Motion Queue	Motion
GMTYPE	Motion Type	Motion
GPATH	Group Path	Motion
GPHASE	Motion Phase	Motion
GRTIME	Remaining Motion Time	Motion
GSEG	Motion Segment	Motion
GSFREE	Free Motion Segments	Motion
GVEC	Group Vector	Motion
GVEL	Group Velocity	Motion
IENA	Interrupt Enable/Disable	System Configuration
IMASK	Interrupt Mask	System Configuration
IN	Digital Inputs	Input and Output
IND	Primary Index Position	Axis State
ISENA	Specific Interrupt Enable/Disable	System Configuration
IST	Index State	Axis State
JERK	Jerk	Motion
KDEC	Kill Deceleration	Motion
M2ARK	Secondary Mark Position	Axis State
MARK	Primary Mark Position	Axis State
MERR	Motor Error	Safety Control
MFF	Master Feed Forward	Axis Configuration
MFLAGS	Motor Flags	Axis Configuration
MPOS	Master Position	Axis State
MST	Motor State	Axis State

Table 19 Alphabetical Listing of All ACSPL+ Variables (page 4 of 6)

Name	Description	Variable Group
NVEL	Minimal Velocity	Motion
ONRATE	Autoroutine Rate	Program Execution Control
OUT	Digital Outputs	Input and Output
PCHARS	Program Size in Characters	Program Execution Control
PE	Non-Critical Position Error	Axis State
PERL	Program Error Line	Program Execution Control
PERR	Program Error	Program Execution Control
PEXL	Program Executed Line	Program Execution Control
PFLAGS	Program Flags	Program Execution Control
PLINES	Program Size in Lines	Program Execution Control
PRATE	Program Rate	Program Execution Control
PST	Program State	Program Execution Control
RACC	Reference Acceleration	Axis State
ROFFS	Reference Offset	Axis State
RPOS	Reference Position	Axis State
RVEL	Reference Velocity	Axis State
RVFIL	Reference Velocity Filter	Axis Configuration
S_DCN	System DC, Number of Samples	Data Collection
S_DCP	System DC, Period	Data Collection
S_ERR	System Error	Safety Control
S_FAULT	System Faults	Safety Control
S_FDEF	System Default Response Mask	Safety Control
S_FLAGS	System Flags	System Configuration
S_FMASK	System Fault Mask	Safety Control
S_SAFIN	System Safety Inputs	Safety Control
S_SAFINI	System Safety Inputs Inversion	Safety Control
S_ST	State of System Data Collection	Data Collection
SAFIN	Safety Inputs	Safety Control
SAFINI	Safety Inputs Inversion	Safety Control
SCCOFFS	Sin-Cos Offset (Cosine)	Axis Configuration
SCSOFFS	Sin-Cos Offset (Sine)	Axis Configuration
SETTLE	Settling Time	Axis Configuration
SLABITS	Absolute Position Bits	Servo-Loop - Miscellaneous
SLAFF	Acceleration Feed Forward	Servo-Loop - Position
SLBIASA	Current Phase A Bias	Servo-Loop - Current
SLBIASB	Current Phase B Bias	Servo-Loop - Current
SLCFIELD	Induction Motor Excitation	Induction Motor
SLCHALL	Hall Shift	Servo-Loop - Commutation

Table 19 Alphabetical Listing of All ACSPL+ Variables (page 5 of 6)

Name	Description	Variable Group
SLCNP	Number of Motor Poles	Servo-Loop - Commutation
SLCOFFS	Commutation Offset	Servo-Loop - Commutation
SLCORG	Commutation Origin	Servo-Loop - Commutation
SLCPA	Phase Advance	Servo-Loop - Commutation
SLCPRD	Commutation Period	Servo-Loop - Commutation
SLCROUT	Commutation Feedback	Servo-Loop - Miscellaneous
SLCSLIP	Induction Motor Slip Factor	Induction Motor
SLDRA	DRA frequency	Servo-Loop - Compensations
SLDRX	Maximum DRA correction	Servo-Loop - Compensations
SLDZMAX	Maximum Dead Zone position	Servo-Loop - Nanomotion
SLDZMIN	Minimum Dead Zone position	Servo-Loop - Nanomotion
SLFRC	Static Friction	Servo-Loop - Compensations
SLFRCD	Dynamic Friction	Servo-Loop - Compensations
SLIKI	Integrator Gain	Servo-Loop - Current
SLIKP	Integrator Proportional Gain	Servo-Loop - Current
SLIOFFS	Current Command Offset	Servo-Loop - Miscellaneous
SLLIMIT	Soft Left Limit	Axis Configuration
SLPBITS	Position Feedback Bits	Servo-Loop - Miscellaneous
SLPKP	Proportional Position Gain	Servo-Loop - Position
SLPMAX	Modulo Axis Upper Limit	Axis Configuration
SLPMIN	Modulo Axis Lower Limit	Axis Configuration
SLPROUT	Position Feedback Routing	Servo-Loop - Miscellaneous
SLVB0DD	Bi-Quad filter damping ratio denominator	Servo-Loop - Filter
SLVB0DF	Bi-Quad filter algorithm denominator	Servo-Loop - Filter
SLVB0ND	Bi-Quad filter damping ratio numerator	Servo-Loop - Filter
SLVB0NF	Bi-Quad filter algorithm numerator	Servo-Loop - Filter
SLVKI	Velocity Integrator Coefficient	Servo-Loop - Velocity
SLVKP	Proportional Velocity Gain	Servo-Loop - Velocity
SLVLI	Integrator Velocity Limit	Servo-Loop - Velocity
SLVNATT	Notch Filter Attenuation	Servo-Loop - Filter
SLVNFRQ	Notch Filter Frequency	Servo-Loop - Filter
SLVNWID	Notch Filter Width	Servo-Loop - Filter
SLVRAT	Velocity Feed Forward Ratio	Servo-Loop - Miscellaneous
SLVROUT	Velocity Feedback	Servo-Loop - Miscellaneous
SLVSOF	Low-Pass Filter Bandwidth	Servo-Loop - Filter
SLVSOFD	Low-Pass Filter Damping	Servo-Loop - Filter
SLZFF	Defines zero velocity feed forward position	Nanomotion
SMNA	Software Monitor Address	Monitoring

Table 19 Alphabetical Listing of All ACSPL+ Variables (page 6 of 6)

Name	Description	Variable Group
SMNV	Software Monitor Value	Monitoring
SRLIMIT	Soft Right Limit	Axis Configuration
STEPF	Stepper Factor	Axis Configuration
STEPW	Stepper Pulse Width	Axis Configuration
SYNV	Synchronous Velocity	Axis Configuration
TARGRAD	Target Radius	Axis Configuration
TCPIP	IP Address for 1st Ethernet	Communication
TCPIP2	IP Address for 2nd Ethernet	Communication
TCPPORT	TCP Port Number	Communication
TIME	Elapsed Time	Monitoring
TPOS	Target Position	Motion
UDPPORT	UDP Port Number	Communication
USAGE	MPU Usage	Monitoring
VEL	Velocity	Motion
VELBRK	Brake Velocity	Motion
XACC	Maximum Acceleration	Axis Configuration
XARRSIZE	Maximum Array Size	Miscellaneous
XCURI	Maximum Current in Idle	Axis Configuration
XCURV	Maximum Current in Moving	Axis Configuration
XRMS	RMS Current Limit	Axis Configuration
XRMST	RMS Current Time Constant	Axis Configuration
XSACC	Maximum Slave Acceleration	Axis Configuration
XVEL	Maximum Velocity	Axis Configuration

3.1 Axis Configuration Variables

The Axis Configuration variables are:

Name	Description
AFLAGS	Axis Flags
BOFFTIME	Brake Deactivation Time
BONTIME	Brake Activation Time
CERRA	Critical Position Error (Accelerating)
CERRI	Critical Position Error (Idle)
CERRV	Critical Position Error (Velocity)
DELI	Delay on Transition to Idle State
DELV	Delay on Transition to Velocity State
EFAC	Encoder Factor
E2FAC	Secondary Encoder Factor
E_FREQ	Encoder Frequency
E_SCMUL	Encoder Sin-Cos Multiplier
E_TYPE	Encoder Type
ENTIME	Enable Time
EOFFS	Axis Encoder Offset
E2OFFS	Axis Encoder Offset for Secondary Encoder
ERRA	Tolerable Error (Accelerating)
ERRI	Tolerable Error (Idle)
ERRV	Tolerable Error (Velocity)
FVFIL	Primary Feedback Velocity Filter
MFF	Master Feed forward
MFLAGS	Motor Flags
RVFIL	Reference Velocity Filter
SCCOFFS	Defines the cosine offset.
SCSOFFS	Defines the sine offset.
SETTLE	Axis Settling Time Parameter
SLCFIELD	Defines magnetic field component of an induction motor
SLCSLIP	Defines the slip constant component of an induction motor
SLLIMIT	Software Left Limit-feedback count down limit
SRLIMIT	Software Right Limit-feedback count up limit
SLPMAX	Specifies the upper limit of modulo axis
SLPMIN	Specifies the lower limit of modulo axis
STEPF	Stepper Factor
STEPW	Stepper Pulse Width
SYNV	Synchronous velocity for **MASTER-SLAVE** motion

Name	Description
TARGRAD	Axis settling target envelope parameter
XACC	Over Acceleration fault parameter
XCURI	Maximum idle motor current
XCURV	Maximum drive current during motion
XRMS	Axis RMS over current fault parameter
XRMST	RMS Current Time Constant
XSACC	Maximum slave axis acceleration
XVEL	Over Velocity fault parameter

3.1.1 AFLAGS

Description

AFLAGS is a 64 member (one for each axis) integer array containing a set of 4 bits.

Syntax

AFLAGS*axis_index.bit_designator = 1|0*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
bit_designator	The meanings of **bit_designator** are given in **Table 20**.meanings of **bit_designator**

Table 20 AFLAGS Bit Description

Bit Name	No.	Description
#NOS	0	No S-Profile
#SEMIS	1	Semi-S-Profile
#AUX	2	Auxiliary Axis
#NOGROUP	3	Disable Group

Tag

3

Comments

Currently, only **AFLAGS**(axis_index).**#NOGROUP** can be set in the variable. **AFLAGS**(axis_index).**#NOGROUP** disables the axis in a group, so that any group or motion command that includes the axis in a group will fail.

Other bits are reserved for future use, and must be set to zero.

Note	
	AFLAGS *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.2 BOFFTIME

Description

BOFFTIME is a 64 member (one for each axis) real array used for specifying the brake deactivation time in milliseconds.

Syntax

BOFFTIME*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.22507e-308 to 1.79769e+308, Default = 100000.

Tag

9

Accessibility

Read-Write

Related ACSPL+ Commands

ENABLE/ENABLEALL - The brake is deactivated automatically when the ENABLE command is executed.

Related ACSPL+ Variables

MFLAGS

COM Library Methods

Enable, EnableM

C Library Functions

acsc_Enable, acsc_EnableM

3.1.3 BONTIME

Description

BONTIME is a 64 member (one for each axis) real array used for specifying the brake activation time in milliseconds.

Syntax

BONTIME*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.22507e-308 to 1.79769e+308, Default = 100000.

Tag

10

Accessibility

Read-Write

Related ACSPL+ Commands

DISABLE, **FCLEAR**, **DISABLEALL**

Note	
	The brake is activated automatically when the **DISABLE** *command is executed.*

Related ACSPL+ Variables

MERR, **FPOS**, **RPOS**

COM Library Methods

Disable, DisableM

C Library Functions

acsc_Disable, acsc_DisableM

3.1.4 CERRA

Description

CERRA is a 64 member (one for each axis) real array, used for defining the Position Error criterion for acceleration/deceleration states.

Syntax

CERRA(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** of each member ranges between 2.22507e-308 and 1.79769e+308, Default = 1000.

Tag

11

Comments

CERRA defines critical position error fault (**FAULT**(axis_index).**#CPE**) criterion when the motor is in acceleration or deceleration motion states.

As a configuration variable, **CERRA** is normally defined by **SPiiPlus MMI Application Studio → Toolbox → Setup → Adjuster** during the setup procedure of the system.

Note	
	CERRA *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

FAULT(axis_index).**#CPE**

CERRI, **CERRV**, **DELI**, **DELV**

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.5 CERRI

Description

CERRI is a 64 member (one for each axis) real array used for defining the critical Position Error when the motor is idle.

Syntax

CERRI(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** of each member ranges between 2.22507e-308 and 1.79769e+308, Default = 1000.

Tag

12

Comments

As a configuration variable, **CERRI** is normally defined by **SPiiPlus MMI Application Studio → Toolbox → Setup → Adjuster** during the setup procedure of the system.

Note	
	CERRI *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

FAULT(axis_index).**#CPE**

CERRA, **CERRV**, **DELI**, **DELV**

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.6 CERRV

Description

CERRV is a 64 member (one for each axis) real array used for defining the critical Position Error when the motor is moving with constant velocity.

Syntax

CERRV(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** of each member ranges between 2.22507e-308 and 1.79769e+308, Default = 1000.

Tag

13

Comments

As a configuration variable, **CERRV** is normally defined by **SPiiPlus MMI Application Studio → Toolbox → Setup → Adjuster** during the setup procedure of the system.

Note	
	CERRV *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

FAULT(axis_index).**#CPE**

CERRA, **CERRI**, **DELI**, **DELV**

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.7 DELI

Description

DELI is a 64 member (one for each axis) real array used for defining the delay of transition to the Idle state.

Syntax

DELI(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** of each member ranges between 2.22507e-308 and 1.79769e+308, Default = 50.

Tag

23

Comments

DELI is defined in milliseconds and applies a delay when the motion state changes from any motion state to idle (**GPHASE**(axis_index) = 0 or 12).

DELI affects the following faults and current limits:

- **FAULT**(axis_index)**.#PE**
- **FAULT**(axis_index)**.#CPE**
- **XCURI**(axis_index)
- **XCURV**(axis_index)

Note	
	DELI *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

ERRA, **ERRI**, **ERRV**, **DELV**, **FAULT**(axis_index).**#CPE**, **FAULT**(axis_index).**#PE**

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.8 DELV

Description

DELV is a 64 member (one for each axis) real array used for defining the delay of transition to the Constant Velocity state.

Syntax

DELV(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** of each member ranges between 2.22507e-308 and 1.79769e+308, Default = 50.

Tag

24

Comments

DELV is defined in msecs and applies a delay when the motion state changes to a constant velocity state (**GPHASE**(axis_index) = 4).

DELV affects the following faults:

- **FAULT**(axis_index)**.#PE**
- **FAULT**(axis_index)**.#CPE**

Note	
	DELV *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

ERRA, **ERRI**, **ERRV**, **DELI**, **FAULT**.#CPE, **FAULT**.#PE

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.9 EFAC

Description

EFAC is a 64 member (one for each axis) real array used for defining a factor between the raw feedback in encoder counts and the **FPOS** value calculated by the controller.

Syntax

EFAC(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** of each member ranges between 1e-15 to 1e+15, Default = 1.

Tag

36

Comments

When reading the feedback position from the SP, the controller executes feedback transform according to the formula:

FPOS = FP*EFAC + EOFFS

where **FPOS** is the controller feedback position in user units, FP is an SP-calculated feedback position in encoder counts, **EFAC** is a user-defined value of the corresponding **EFAC** factor, and **EOFFS** represents an offset.

As a configuration variable, **EFAC** is normally defined by **SPiiPlus MMI Application Studio → Toolbox → Setup → Adjuster** during the setup procedure of the system.

Accessibility

Read-Write

Related ACSPL+ Variables

All standard variables that are based on position units.

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.10 E2FAC

Description

E2FAC is a 64 member (one for each axis) real array used for defining a factor between the secondary raw feedback in encoder counts and the **F2POS** value calculated by the controller.

Syntax

E2FAC(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** of each member ranges between 1e-15 to 1e+15, Default = 1.

Tag

32

Comments

When reading the secondary feedback position from the SP, the controller executes feedback transform according to the formula:

F2POS = FP2*E2FAC + E2OFFS

where F2POS is the secondary controller feedback position in user units, FP2 is an SP-calculated secondary feedback position in encoder counts, E2FAC is a user-defined value of the corresponding E2FAC factor, and E2OFFS represents an offset.

As a configuration variable, **EFAC** is normally defined by **SPiiPlus MMI Application Studio → Toolbox → Setup → Adjuster** during the setup procedure of the system.

Accessibility

Read-Write

Related ACSPL+ Variables

F2POS, EOFFS

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.11 E_FREQ

Description

E_FREQ is a 64 member (one for each axis) integer array used for defining the maximum encoder pulse frequency (in MHz).

Syntax

E_FREQ(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** of each member can be one of: ❑ 2 (shown as 2MHz but practically is 2.5MHz, default for an analog Sin-Cos encoder) ❑ 20 (default for a digital encoder) ❑ 60

Tag

27

Comments

The encoder is represented in the controller as a synchronous state machine that is activated by a clock, with programmable frequency in the SPii processor.

E_FREQ provides three optional clock rates that define the maximum encoder pulse frequency (in MHz) measured after an internal 4x multiplication. **E_FREQ** is relevant only for the 0, 1, 4, and 5 axes. A change for 0 also applies for 2, a change for 1 also applies for 3, etc.

In general, using a higher **E_FREQ** enables to read a higher rate of encoder input. However, the electrical noise immunity is reduced and Encoder Error **FAULT** might occur. Per case, it is recommended to use the lowest possible **E_FREQ** that does not generate an Encoder Error FAULT. In case of an Encoder Error, do **FCLEAR** (axis_index) and try a higher **E_FREQ** value.

For more information, see the Encoder Input Clock section in the *SPiiPlus Setup Guide*.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.12 E_TYPE

Description

E_TYPE is a 64 member (one for each axis) integer array used for defining the encoder type.

Syntax

E_TYPE(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** of each member can be one of: ❑ 0 - Up-down counter ❑ 1 - Clock direction counter ❑ 3 - Quadrature encoder (Default) ❑ 4 - Sin-Cos encoder multiplier

Tag

29

Comments

The most common encoder type is quadrature, which corresponds to the default value 3.

As a configuration variable, the variable can be changed only if the controller is in configuration mode.

Accessibility

Read-Write

Related ACSPL+ Variables

None

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.13 EOFFS

Description

EOFFS is a 64 member (one for each axis) integer array used for defining the offset between the raw feedback from the encoder counts and the **FPOS** value calculated by the controller.

Syntax

EOFFS(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** can be any integer.

Tag

38

Comments

EOFFS provides the offset between the raw feedback in encoder counts and the **FPOS** value calculated by the controller. The value of **EOFFS** changes when the set command defines a new origin for an axis.

When reading the feedback position from the SP, the controller executes feedback transform according to the formula:

FPOS = FP*EFAC + EOFFS

where FPOS is the controller feedback position in user units, **FP** is an SP-calculated feedback position in encoder counts, **EFAC** is a user-defined value of the corresponding EFAC factor, and EOFFS represents an offset.

Accessibility

Read-Only

Related ACSPL+ Variables

EFAC, **FPOS**

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.1.14 E2OFFS

Description

E2OFFS is a 64 member (one for each axis) integer array used for defining the offset between the raw feedback from the secondary encoder counts and the **FPOS** value calculated by the controller.

Syntax

E2OFFS(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** can be any integer.

Tag

34

Comments

E2OFFS provides the offset between the raw feedback from the secondary encoder (in encoder counts) and the **F2POS** value calculated by the controller. The value of **E2OFFS** changes when the set command defines a new origin for the axis's secondary feedback.

When reading the secondary feedback position from the SP, the controller executes feedback transform according to the formula:

F2POS = FP2*E2FAC + E2OFFS

where **F2POS** is the secondary controller feedback position in user units, **FP2** is an SP-calculated secondary feedback position in encoder counts, **E2FAC** is a user-defined value of the corresponding **E2FAC** factor, and **E2OFFS** represents an offset.

Accessibility

Read-Only

Related ACSPL+ Variables

E2FAC, **F2POS**

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.1.15 E_SCMUL

Description

E_SCMUL is a 64 member (one for each axis) integer array used for specifying the Sin-Cos multiplication factor for the encoder.

Syntax

E_SCMUL(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2 to 16, Default = 10.

Tag

28

Comments

E_SCMUL specifies the Sin-Cos multiplication factor as a power of 2. The maximum value of 16 corresponds to a multiplication of 65536 = 2^{16}. The minimum value of 2 corresponds to a multiplication of 4 = 2^2.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.16 ENTIME

Description

ENTIME is a 64 member (one for each axis) real array used for controlling the execution of **ENABLE/ENABLEALL**.

Syntax

ENTIME(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.22507e-308 to 1.79769e+308, Default = 50.

Tag

37

Comments

Since the drive enable process is relatively long, (50 - 100msec) **ENTIME** defines a time duration in msec between **ENABLE/ENABLEALL** and the moment the controller considers the drive as enabled and all faults as **FAULT(axis_index).#PE**, **FAULT(axis_index).#CPE**, **FAULT**(axis_index).#DRIVE are triggered.

Exact usage of the variable depends on the flag bit **MFLAGS**(axis_index)**.#ENMODE** (bit 19). See **MFLAGS**.

Accessibility

Read-Write

Related ACSPL+ Commands

ENABLE/ENABLEALL

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.17 ERRA

Description

ERRA is a 64 member (one for each axis) real array used for defining the Position Error criterion for Acceleration/Deceleration states.

Syntax

ERRA(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.22507e-308 to 1.79769e+308, Default = 100.

Tag

39

Comments

ERRA defines the maximum tolerable position error (**FAULT**(axis_index).**#PE**) when the motor is moving with acceleration.

As a configuration variable, **ERRA** is normally defined by **SPiiPlus MMI Application Studio → Toolbox → Setup → Adjuster** during the setup procedure of the system.

Note	
	ERRA *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

FAULT(axis_index).**#PE**, **FDEF**

ERRI, **ERRV**, **DELI**, **DELV**

COM Library Methods

ReadVariable, WriteVariable,

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.18 ERRI

Description

ERRI is a 64 member (one for each axis) real array used for defining the maximum tolerable Position Error (**FAULT**(axis_index).**#PE**) when the motor is idle.

Syntax

ERRI(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.22507e-308 to 1.79769e+308, Default = 100.

Tag

40

Comments

As a configuration variable, **ERRI** is normally defined by **SPiiPlus MMI Application Studio → Toolbox → Setup → Adjuster** during the setup procedure of the system.

Note	
	ERRI*values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

FAULT(axis_index)**.#PE**

ERRA, **ERRV**, **DELI**, **DELV**

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.19 ERRV

Description

ERRV is a 64 member (one for each axis) real array used for defining the maximum tolerable Position Error (**FAULT**(axis_index)**.#PE**) when the axis is moving with constant velocity.

Syntax

ERRV(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.22507e-308 to 1.79769e+308, Default = 100.

Tag

41

Comments

As a configuration variable, **ERRV** is normally defined by **SPiiPlus MMI Application Studio → Toolbox → Setup → Adjuster** during the setup procedure of the system.

Note	
	ERRV *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

FAULT(*axis_index*)**.#PE**

ERRA, **ERRI**, **DELI**, **DELV**

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.20 FVFIL

Description

FVFIL is a 64 member (one for each axis) real array used for setting the intensity (in %) of the filter that the controller uses when calculating **FVEL**.

Syntax

FVFIL(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 0 to 100, Default = 30.

Tag

54

Comments

FVFIL = 0 corresponds to no filtering. In this case the controller calculates **FVEL** as the derivative of the **FPOS** variable:

$\Delta = (FPOS_n - FPOS_{n-1})*K$

$FVEL_n = \Delta$

where K is a scaling factor that reduces **FVEL** to user units per second.

As the **FPOS** value is supplied by a discrete physical sensor, the **FVEL** value calculated without filtering contains a considerable amount of noise.

Non-zero value of **FVFIL** provides additional filtering in the **FVEL** calculation according to the formula:

$FVEL_n = \Delta*((1 - FVFIL)/100) + FVEL_{n-1}*(FVFIL/100))$

Accessibility

Read-Write

Related ACSPL+ Variables

FVEL

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.21 MFF

Description

MFF is a 64 member (one for each axis) real array used for specifying the feed forward time, in milliseconds, for **MPOS** calculations.

Syntax

MFF(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 0 to 10, Default = 2.

Tag

87

Comments

The controller calculates the **MPOS** value according to a formula supplied by the **MASTER** command. A non-zero **MFF** value provides additional extrapolation of the calculated value to the predicted value at the current time plus **MFF**. The purpose is to compensate delay introduced by the controller and the external circuits.

The default value of MFF depends on the controller model so that it compensates the delay introduced by the controller itself. Increase the MFF value if you want to compensate additional delay introduced by sensor or other circuits.

Note	
	MFF *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Configuration

Related ACSPL+ Commands

MASTER

Related ACSPL+ Variables

ENTIME

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.22 MFLAGS

Description

MFLAGS is a 64 member (one for each axis) integer array containing a set of bits used for configuring the motor.

Syntax

MFLAGS(*axis_index).bit_designator = 0|1*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
bit_designator	The **MFLAGS** bit designators are given in **Table 21.**

Table 21 MFLAGS Bit Designators (page 1 of 4)

Name	No.	Description
#DUMMY	0	0 (default) = The axis is defined as non-dummy. 1 = The axis is defined as dummy. A dummy axis is an inactive axis which is not connected to a drive and a motor. When an axis is defined as a dummy the following apply: ❑ In SPiiPlus controllers, excluding Control Modules, the two analog outputs of the axis can be used as General Purpose outputs. ❑ In all SPiiPlus products: all faults of the axis are disabled. ❑ In all SPiiPlus products **ENABLE/ENABLEALL** is disabled.
#OPEN	1	0 (default): Closed-loop control - for servo motors only. 1: Open-loop control. In open loop control the user defines the value of the command/s to the drive with variable **DCOM.**
#HOME	3	0 (default): Axis homing procedure not done. 1: Axis homing procedure done. This bit can be changed by a user program to 1 (**MFLAGS**(*axis_index*).**#HOME** = 1) following a successful homing process. The controller clears the bit during power-up, but then neither changes the bit nor verifies its value. Based on this bit the user can implement any logic in his program application.
#STEPPER	4	0 (default): Axis is defined as a servo motor. 1: Axis is defined as a pulse-direction stepper motor in open loop.

Table 21 MFLAGS Bit Designators (page 2 of 4)

Name	No.	Description
#ENCLOOP	5	0 (default): Stepper feedback loop is not active. 1: The axis provides a stepper feedback loop. In this case the **FPOS**(axis_index) variable will indicate the number of pulses that were sent to the pulse-direction stepper drive and not the encoder counts - if an encoder is connected and **#STEPENC** (bit 6 = 1). **#ENCLOOP** is effective only if **#STEPPER** (bit 4) = 1.
#STEPENC	6	0 (default): Stepper encoder feedback loop is not active. 1: The controller provides an encoder feedback to the pulse-direction stepper. In this case, the **FPOS**(axis_index) variable will indicate the quadrature encoder counts, and not the stepper pulse-direction pulses if **#ENCLOOP** (bit 5 = 1). **#STEPENC** is effective only if **#STEPPER** (bit 4) = 1. The encoder feedback is used for monitoring the axis position by a user application, and does not affect the open loop control of the stepper motor.
#NANO	7	0 (default): Defines the axis as a servo or stepper motor. 1: Defines the axis as a Nanomotion piezo ceramic motor.
#BRUSHL	8	0 (default): The motor is a non-DC brushless type or the amplifier provides commutation itself and uses one ±10V input from the controller. 1: The controller provides commutation for the DC brushless motor. The bit must be set only if the controller is connected to a three-phase amplifier with two-phase input commands.
#BRUSHOK	9	**#BRUSHOK** is applicable only for DC Brushless motors when **#BRUSHL.8** =1. 0 (default): DC brushless motor is not commutated. 1: DC brushless motor is commutated. After power-up the controller clears the bit. The controller automatically sets this bit =1 only after a successful commutation or auto-commutation processes.
#DBRAKE	11	0 (default): Dynamic brake is disabled 1: The controller will apply dynamic braking to stop the motor when the axis is disabled and the feedback velocity is less than the predefined value of **VELBRK**. See the *SPiiPlus Setup Guide*, Dynamic Brake for a complete explanation.
#INVENC	12	0 (default): Encoder counting is non-inverted. 1: The controller inverts the direction of encoder counting. This does not affect the motion direction.
#INVDOUT	13	0 (default): Drive output command/s are not inverted. 1: The controller inverts the drive output command/s. This effectively inverts the direction of the motion and the sign of the feedback.
#NOTCH	14	0 (default): Notch filter is disabled. 1: Notch filter is active
#NOFILT	15	0 (default): The control algorithm includes a second-order filter specified by bandwidth **SLVSOF**. 1: The control algorithm by-passes the second-order filter.

Table 21 MFLAGS Bit Designators (page 3 of 4)

Name	No.	Description
#DEFCON	17	0: **CONNECT** is allowed. See **CONNECT**. 1 (default): The controller applies the default connection between **APOS** and **RPOS** (**RPOS**=**APOS**). While **#DEFCON** = 1, the controller does not accept **CONNECT** for the axis. The bit is reset to 1 every time the controller is restarted.
#ENMOD	19	0: Enable time is defined by **ENTIME**, or when the drive switches its drive alarm signal to the inactive state - whichever comes first. If the signal remains active more than **ENTIME** milliseconds, **ENABLE/ENABLEALL** fails. 1 (default): The value of **ENTIME** defines the enable time.
#DUALLOOP	20	0 (default): The control algorithm implements a regular single-loop scheme obtaining the position and velocity feedback from the same feedback sensor as defined by **SLPROUT**. 1: The control algorithm implements a dual-loop scheme.
#LINEAR	21	0 (default): The axis is defined as rotary. 1: The axis is defined as linear.
#BRAKE	23	0 (default): The controller does not provide mechanical brake control. 1: The brake is deactivated when the motor is enabled and activated when the motor is disabled. For detailed description, see commands **ENABLE/ENABLEALL** and **DISABLE**, and variables **BOFFTIME**, **BONTIME** variables.
#HSSI	24	0 (default): Direct drive connection. 1: Remote drive connection via HSSI.
#COG	25	0 (default): Cogging compensation disabled 1: Cogging compensation enabled.
#PI	26	0 (default): The controller provides a current command to the external drive (when the external drive provides current loop control). 1: The controller provides a velocity command to the drive when the external drive provides velocity and current loops control.
#HALL	27	*Note:* **#HALL** *applies only to DC brushless motors when* **#BRUSHL** *(bit 8) =1.* 0 (default): Commutation is not based on Hall signals. 1: Commutation is based on Hall signals.
#INVHALL	28	*Note:* **#INVHALL** *applies only for DC brushless motors when* **#BRUSHL** *(bit 8) =1.* 0 (default): Motor Hall signals are non-inverted. 1: Motor Hall signals are inverted.

Table 21 MFLAGS Bit Designators (page 4 of 4)

Name	No.	Description
#MODULO	29	0 (default): Axis is defined as non-modulo. 1: Axis is defined as **MODULO**. In **MODULO** mode, physically, the motion of the axis is not limited, but each time **RPOS** goes out of the defined **SLPMIN**...**SLPMAX** range, the controller brings **RPOS** into range by changing the internal offset **EOFFS**. See **SLPMAX** and **SLPMIN**.
#USER1	30	The functionality of this bit can be defined by the user. 0 (default): Functionality not defined 1: User defined functionality
#USER2	31	The functionality of this bit can be defined by the user. 0 (default): Functionality not defined 1: User defined functionality

Tag

88

Comments

MFLAGS is typically configured using the **SPiiPlus MMI Application Studio → Toolbox → Setup → Adjuster** when setting the Dynamic Brake.

Use direct bit assignment for on-the-fly changes, for example, from closed-loop operation to the open-loop and vice versa.

Note	
	MFLAGS *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Configuration

Related ACSPL+ Commands

CONNECT, **ENABLE/ENABLEALL**, **GETCONF**

Related ACSPL+ Variables

SLPROUT, **APOS**, **RPOS**

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.23 RVFIL

Description

RVFIL is a 64 member (one for each axis) real array used for specifying the power of the filter that the controller uses to calculate **RVEL**.

Syntax

RVFIL(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** is a percent ranging from 0 to 100, Default = 0.

Tag

110

Comments

The value is specified as a percent where 0 means no filtering. In this case the controller calculates **RVEL** as the first difference of **RPOS** as follows:

$\Delta = (RPOS - RPOS_{n-1}) * K$

$RVEL_n = \Delta$

where K is a scaling factor that translates **RVEL** to position units per second.

A non-zero value for **RVFIL** provides additional filtering in the **RVEL** calculation according to the formula:

$RVEL_n = \Delta * ((1- (RVFIL/100)) + (RVEL_{n-1} * (RVFIL/100))$

The default value of **RVFIL** is zero. No filtering is required as long as the axis motion is not a **MASTER-SLAVE** motion. In this case **RPOS** is a calculated value and the first difference provides a smooth approximation of velocity.

If an axis is involved in a **MASTER-SLAVE** motion, **RPOS** usually contains a signal from a discrete physical sensor that causes a certain amount of noise in the first difference. Increase the value of **RVFIL** if a smoother approximation is required.

Note	
	RVFIL *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Configuration

Related ACSPL+ Commands

MASTER

Related ACSPL+ Variables

RVEL

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.24 SCCOFFS

Description

SCSOFFS is a 64 member (one for each axis) real array used for defining the Sin-Cos Cosine offset.

Syntax

SCCOFFS*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** designates the offset ranging from -32766 to 32766.

Tag

203

Comments

The digital range corresponds to ±0.5 Volts. This number is used to modify the offset of the Sin-Cos encoder signal related to the axis. The ratio between this number and the offset is 9.6.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadInteger, acsc_WriteInteger

3.1.25 SCSOFFS

Description

SCSOFFS is a 64 member (one for each axis) real array used for defining the Sin-Cosine offset.

Syntax

SCSOFFS*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** designates the offset ranging from -32766 to 32766.

Tag

202

Comments

The digital range corresponds to ±0.5 Volts. This number is used to modify the offset of the Sin-Cos encoder signal related to the axis. The ratio between this number and the offset is 9.6.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadInteger, acsc_WriteInteger

3.1.26 SETTLE

Description

SETTLE is a 64 member (one for each axis) real array used for setting the Settling Time, it controls **MST**(axis_index)**.#INPOS**.

Syntax

SETTLE(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 0 to 1.79769e+308, Default = 0.

Tag

123

Comments

When the motor is not moving, the controller compares the position error (**PE** value) and the target envelope (**TARGRAD** value) every MPU cycle. **#INPOS** is raised when **PE** drops to within the range (-TARGRAD, +TARGRAD) and remains within the range for a period of time equal or greater than **SETTLE**.

If the motor starts to move or **PE** goes out of the range, **MST**(axis_index)**.#INPOS** is cleared.

Note	
	SETTLE *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Configuration

Related ACSPL+ Variables

TARGRAD, **MST**, **PE**

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.27 SLLIMIT

Description

SLLIMIT is a 64 member (one for each axis) real array used for defining the minimum allowed Left position for the motor.

Syntax

SLLIMIT(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from -1.79769e+308 to 1.79769e+308, Default = 2e+014.

Tag

124

Comments

If reference position **RPOS** is less than this value, a software Left Limit fault results and bit **FAULT**(axis_index)**.#SLL** is = 1.

Note	
	SLLIMIT *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

RPOS, **FAULT**(axis_index).**#SLL**, **SRLIMIT**

COM Library Methods

ReadVariable, WriteVariable, GetFault

C Library Functions

acsc_ReadVariable, acsc_WriteVariable, acsc_GetFault

3.1.28 SLPMAX

Description

SLPMAX is a 64 member (one for each axis) real array used for storing the maximum range of a modulo axis.

Syntax

SLPMAX(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from -1.79769e+308 to 1.79769e+308, Default = 8000.

Tag

194

Comments

SLPMAX stores the maximum range of a modulo axis, see **MFLAGS**.**#MODULO** (bit 29). **SLPMAX** can be changed only when the motor is disabled.

Accessibility

Read-Write

Related ACSPL+ Variables

SLPMIN, **EOFFS**

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.29 SLPMIN

Description

SLPMIN is a 64 member (one for each axis) real array used for storing the minimum range of a modulo axis.

Syntax

SLPMIN(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from -1.79769e+308 to 1.79769e+308, Default = 0.

Tag

195

Comments

SLPMIN stores the minimum range of a modulo axis, see **MFLAGS**.**#MODULO** (bit 29). **SLPMIN** can be changed only when the motor is disabled.

Accessibility

Read-Write

Related ACSPL+ Variables

SLPMAX, **EOFFS**

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.30 SRLIMIT

Description

SRLIMIT is a 64 member (one for each axis) real array used for defining the minimum allowed Right position for the motor.

Syntax

SRLIMIT(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from -1.79769e+308 to 1.79769e+308, Default = 2e+014.

Tag

128

Comments

If reference position **RPOS** is less than this value, a software Right Limit fault results and **FAULT**(axis_index)**.#SRL** is = 1.

Note	
	SRLIMIT *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

RPOS, **FAULT**(axis_index)**.#SRL**, **SLLIMIT**

COM Library Methods

ReadVariable, WriteVariable, GetFault

C Library Functions

acsc_ReadVariable, acsc_WriteVariable, acsc_GetFault

3.1.31 STEPF

Description

STEPF is a 64 member (one for each axis) real array used for defining the ratio between one stepper pulse and user units. See the *SPiiPlus Setup Guide*, Stepper Drive section for more information.

Syntax

STEPF(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 1e-015 to 1e+015, Default = 1.

Tag

129

Comments

STEPF = 1 (default) means that motion is performed in motor steps. For example,

PTP/R 0,320

will move the motor by 320 steps from the current position.

If another unit is required for motion programming, the user must configure an appropriate value for **STEPF**. For example, a controlled plant provides a gear ratio of 500 motor pulses per inch. If the motion programming unit must be provided in inches, the configured **STEPF** value must be 0.002 (1/500).

Note	
	STEPF *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.32 STEPW

Description

STEPW is a 64 member (one for each axis) real array used for defining the pulse width, in milliseconds, for the stepper motor. See the *SPiiPlus Setup Guide*, Stepper Drive section for more information.

Syntax

STEPW(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.5e-005 to 0.051, Default = 0.001.

Tag

130

Comments

The value defines the width of the pulses generated on the pulse output for stepper control. The value is limited from 25e-6 (25 nanoseconds) to 0.051 (51 microseconds).

Note	
	STEPW *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.33 SYNV

Description

SYNV is a 64 member (one for each axis) real array used for defining the criteria for the allowed difference between **MASTER** and **SLAVE** velocities during slave-master axis synchronization.

Syntax

SYNV(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.22507e-308, 1.79769e+308, Default = 1.

Tag

131

Comments

Synchronous velocities of master and slaved axes in velocity lock mode are indicated by the slaved axis **AST**(axis_index)**.#VELLOCK** (bit 8).

Synchronous positioning of master and slaved axes in position lock mode is indicated by the slaved axis **AST**(axis_index)**.#POSLOCK** (bit 9).

MASTER-**SLAVE** motion always begins out of synchronization. The controller tries to achieve slave-master synchronization where the slave pursues the master in order to synchronize their velocities. The velocities are considered synchronized if the absolute difference is less than **SYNV**.

Note	
	SYNV *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Related ACSPL+ Commands

MASTER, **SLAVE**

Related ACSPL+ Variables

XSACC

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.34 TARGRAD

Description

TARGRAD is a 64 member (one for each axis) real array used for defining the parameters for **MST**(axis_index).**#INPOS**.

Syntax

TARGRAD(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.22507e-308, 1.79769e+308, Default = 1.

Tag

132

Comments

When the motor is enabled but in a standstill position, the controller compares **PE** to the target envelope (**TARGRAD**) each MPU cycle. **#INPOS** = 1 when **PE** moves into the defined range (-TARGRAD, +TARGRAD) and remains within that range for a period of time equal or greater than defined by **SETTLE**.

If the motor starts to move or goes out of range, **#INPOS** is cleared.

Note	
	TARGRAD *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Related ACSPL+ Variables

SETTLE, **MST**(axis_index)**.INPOS**, **PE**

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.35 XACC

Description

XACC is a 64 member (one for each axis) real array used for defining the maximum allowed acceleration for the motor.

Syntax

XACC(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.22507e-308 to 1.79769e+308, Default = 1e+007.

Tag

141

Comments

If the reference acceleration **RACC** exceeds this value, the Acceleration Limit fault is activated and bit **#AL** is set in variable **FAULT**.

Note	
	XACC *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Related ACSPL+ Variables

FAULT(axis_index)**.#AL**, **RACC**

COM Library Methods

GetFault, ReadVariable, WriteVariable

C Library Functions

acsc_GetFault, acsc_ReadVariable, acsc_WriteVariable

3.1.36 XCURI

Description

XCURI is a 64 member (one for each axis) real array used for limiting the drive output when the motor is enabled but in standstill position.

Syntax

XCURI(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 0 to 100, Default = 50.

Tag

142

Comments

XCURI is defined as a percentage of the maximum output voltage. For example, in SPiiPlus PCI-4/8, the maximum output voltage is ±10V. Setting **XCURI** to 50 will limit the drive output to ±5V when the motor is idle.

Note	
	XCURI *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Related ACSPL+ Variables

FAULT(axis_index)**.#CL**, **XRMS**, **XCURV**

COM Library Methods

GetFault, ReadVariable, WriteVariable

C Library Functions

acsc_GetFault, acsc_ReadVariable, acsc_WriteVariable

3.1.37 XCURV

Description

XCURV is a 64 member (one for each axis) real array used for limiting the drive output when the motor is moving.

Syntax

XCURV(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 0 to 100, Default = 50.

Tag

143

Comments

XCURV is defined as a percentage of the maximum output voltage. For example, in SPiiPlus PCI-4/8, the maximum output voltage is ±10V. Setting **XCURV** to 50 will limit the drive output to ±5V when the motor is moving.

Note	
	XCURV *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Related ACSPL+ Variables

FAULT(axis_index)**.#CL**, **XRMS**, **XCURV**

COM Library Methods

GetFault, ReadVariable, WriteVariable

C Library Functions

acsc_GetFault, acsc_ReadVariable, acsc_WriteVariable

3.1.38 XRMS

Description

XRMS is a 64 member (one for each axis) real array used for setting the maximum allowable rms current.

Syntax

XRMS(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 0 to 100, Default = 50.

Tag

144

Comments

The SP program calculates **RMS** of the corresponding controller drive output. If the calculated value exceeds the **XRMS** value, an overcurrent fault occurs. **XRMS** is defined as a percentage of the maximum output voltage.

Accessibility

Configuration

Related ACSPL+ Variables

FAULT(axis_index)**.#CL**, **XCURI**, **XCURV**

COM Library Methods

GetFault, ReadVariable, WriteVariable

C Library Functions

acsc_GetFault, acsc_ReadVariable, acsc_WriteVariable

3.1.39 XRMST

Description

XRMST is a 64 member (one for each axis) real array used for setting the time constant in milliseconds for the **XRMS** to activate the overcurrent protection. For calculation of **XRMS** activation time, see *SPiiPlus Setup Guide*.

Syntax

XRMST(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 200 to 3260, Default = 3230.

Tag

145

Accessibility

Configuration

Related ACSPL+ Variables

FAULT(axis_index)**.#CL**, **XCURI**, **XCURV**, **XRMS**

COM Library Methods

GetFault, ReadVariable, WriteVariable

C Library Functions

acsc_GetFault, acsc_ReadVariable, acsc_WriteVariable

3.1.40 XSACC

Description

XSACC is a 64 member (one for each axis) real array used for defining the maximum allowed slave acceleration in **MASTER-SLAVE** motion.

Syntax

XSACC(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.22507e-308 to 1.79769e+308, Default = 1e+007.

Tag

146

Comments

When a slave is synchronized to a master, the controller verifies the slave acceleration against the **XSACC** value each MPU cycle. If the slave acceleration exceeds **XSACC**, the motion falls out of synchronization. The controller tries to regain synchronism by having the slave pursue the master with the maximum allowed motion parameters.

Accessibility

Configuration

Related ACSPL+ Commands

MASTER, **SLAVE**

Related ACSPL+ Variables

SYNV

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.1.41 XVEL

Description

XVEL is a 64 member (one for each axis) real array used for defining the maximum allowed velocity for the axis.

Syntax

XVEL(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.22507e-308 to 1.79769e+308, Default = 2e+006.

Tag

147

Comments

If **RVEL** feedback velocity exceeds **XVEL**, **FAULT**(axis_index)**.#VL** = 1.

Note	
	Trying to adjust the position and velocity loops when **XVEL** *is not correctly set will produce poor results. Verify that* **XVEL** *is correctly defined to fit the application and other requirements before adjusting the loops.*

Related ACSPL+ Variables

FAULT(axis_index).#VL, **RVEL**, **FVEL**

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.2 Axis State Variables

The Axis State variables are:

Name	Description
APOS	Axis Position
AST	Axis State
DAPOS	Delayed Axis Position
FACC	Feedback Acceleration
FPOS	Feedback Position
F2POS	Secondary Feedback Position
FVEL	Feedback Velocity
F2VEL	Secondary Feedback Velocity
IND	Index Position
IST	Index State
MARK	Mark Position
M2ARK	Secondary Mark Position
MPOS	Master Position
MST	Motor State
PE	Position Error
RACC	Reference Acceleration
ROFFS	Reads the offset calculated by the controller in the connect formula.
RPOS	Reference Position
RVEL	Reference Velocity

3.2.1 APOS

Description

APOS is a 64 member (one for each axis) real array used for defining the current reference value for the axis in user-defined units.

Syntax

See **SET**

Tag

6

Comments

APOS is updated each MPU cycle if a motion that involves the axis is in progress.

If the corresponding motor has a default connection (**MFLAGS**(axis_index).**#DEFCON =1**), **APOS** = **RPOS**.

Accessibility

Read-Only - Can be changed using **SET**.

Related ACSPL+ Commands

MASTER-SLAVE

Related ACSPL+ Variables

MPOS

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.2.2 AST

Description

AST is a 64 member (one for each axis) integer array containing a set of bits used for displaying the current axis state.

Syntax

[command] **AST**(*axis_index).bit_designator*

Arguments

command	Typical commands are **DISP** and the like.
axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
bit_designator	A description of the AST bit designators is given in **Table 22**.

Table 22 AST Bit Descriptions (page 1 of 2)

Bit Name	No.	Description
#LEAD	0	1 = axis is leading in a group
#PEG	2	1 = PEG is in progress
#DC	3	1 = Axis data collection is in progress
#PEGREADY	4	1 = all values are loaded and the Incremental/Random PEG is ready to respond to movement
#MOVE	5	1 = Axis is involved in a motion
#ACC	6	1 = Axis in accelerating motion state
#BUILDUP	7	1 = Segments build-up

Table 22 AST Bit Descriptions (page 2 of 2)

Bit Name	No.	Description
#VELLOCK	8	1 = Slave is synchronized to master in velocity lock mode - slave velocity strictly follows the master velocity.
#POSLOCK	9	1 = Slave is synchronized to master in position lock mode - slave position strictly follows the master position.
#TRIGGER	11	1 = Produces an interrupt to the host application, enabled by IENA.26

Tag

7

Accessibility

Read-Only

Related ACSPL+ Commands

MASTER-SLAVE

Related ACSPL+ Variables

MST

COM Library Methods

GetAxisState

C Library Functions

acsc_GetAxisState

3.2.3 DAPOS

Description

DAPOS is a 64 member (one for each axis) integer array. **DAPOS** reads the delayed Axis Position value which is synchronized with **RPOS** and **FPOS**.

Syntax

DAPOS is activated as part of the SPiiPlus MMI Application Studio **Scope**.

Tag

18

Comments

Use **DAPOS** only to view the axis position in the **Scope** when comparing the axis position to the **RPOS**.

In the SPiiPlus, **APOS** (axis position) is not synchronized with **RPOS** and **FPOS** and are characterized by a few msec delay.

When implementing a non-default **CONNECT**, it may be necessary to monitor **APOS** versus **RPOS** with the Scope.

Accessibility

Read-Only

Related ACSPL+ Commands

CONNECT

Related ACSPL+ Variables

APOS, **RPOS**

3.2.4 F2POS

Description

F2POS is a 64 member (one for each axis) real array.

F2POS defines the current secondary feedback value for the motor in user-defined units.

Tag

44

Comments

The user can shift the origin of secondary feedback position using **SET**.

The user can select the units of secondary feedback position by setting the **E2FAC** variable.

The application needs to explicitly clear **IST**(axis_index)**.#IND2** in order to resume the latching logic.

Accessibility

Read-Only - Can be changed by **SET**.

Related ACSPL+ Commands

SET

Related ACSPL+ Variables

IST

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.2.5 FACC

Description

FACC is a 64 member (one for each axis) real array.

FACC defines the feedback acceleration value of the axis.

Tag

46

Accessibility

Read-Only

Related ACSPL+ Variables

FVEL

COM Library Methods

GetAcceleration

C Library Functions

acsc_GetAcceleration

3.2.6 FPOS

Description

FPOS is a 64 member (one for each axis) real array.

FPOS defines the current feedback position for the motor.

Tag

52

Comments

1. The user can shift the origin of feedback position using **SET**.
2. The user can select the units of feedback position by setting the **EFAC** variable.

Accessibility

Read-Only

Related ACSPL+ Commands

SET

Related ACSPL+ Variables

APOS, **RPOS**

COM Library Methods

GetFPosition, SetFPosition

C Library Functions

acsc_GetFPosition, acscSetFPosition

3.2.7 FVEL

Description

FVEL is a 64 member (one for each axis) real array.

FVEL stores the measured velocity.

Tag

53

Accessibility

Read-Only

Related ACSPL+ Variables

FVFIL, **RVEL**, **XVEL**

COM Library Methods

GetFVelocity

C Library Functions

acsc_GetFVelocity

3.2.8 F2VEL

Description

F2VEL is a 64 member (one for each axis) real array.

FVEL stores the measured secondary velocity.

Tag

45

Accessibility

Read-Only

Related ACSPL+ Variables

FVFIL, **RVEL**, **XVEL**, **FVEL**

COM Library Methods

GetFVelocity

C Library Functions

acsc_GetFVelocity

3.2.9 IND

Description

IND is a 64 member (one for each axis) real array.

IND stores the position of the last encountered encoder index in user-defined units.The variable operates in connection with **IST**(axis_index).**#IND**.

Tag

72

Comments

After power-up, **IST**(axis_index).**#IND** is reset and the value of **IND** is undefined because an index capture has not yet occurred. When the motor encounters an encoder index, **IST**(axis_index).**#IND** is raised and the current **FPOS** position is latched to **IND**.

Subsequent index values are ignored as long as **#IND** remains raised.

To resume the latching logic **IST**(axis_index).**#IND** must be explicitly cleared by the command **IST**(axis_index).**#IND=0**.

Accessibility

Read-Only

Related ACSPL+ Variables

IST, **FPOS**

COM Library Methods

GetIndexState, ResetIndexState

C Library Functions

acsc_GetIndexState, acsc_ResetIndexState

3.2.10 IST

Description

IST is a 64 member (one for each axis) integer array containing a set of bits that indicate the state of the **INDEX** and the **MARK** variables.

Syntax

IST(*axis_index).bit_designator = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
bit_designator	The **IST** has four bit designators: ❑ **#IND** (bit 0) - Primary encoder index ❑ **#IND2** (bit 1) - Secondary encoder index ❑ **#MARK** (bit 2) - Mark 1 ❑ **#MARK2** (bit 3) - Mark 2
value	**value** can be zero or non-zero.

Tag

79

Comments

IST contains a set of bits that indicate the state of the **INDEX** and the **MARK** variables as follows:

The controller processes index/mark signals as follows:

When an index/mark signal is encountered for the first time, the controller latches **FPOS** or **F2POS** to one of the variables **IND**, **MARK**, **M2ARK** and sets the corresponding **IST** bit = 1.

As long as an **IST** bit is raised, the controller does not latch another value to the corresponding variable.

To resume the latching logic, the user application must explicitly reset the corresponding **IST** bit = 0.

Accessibility

Read-Write

Related ACSPL+ Variables

FPOS,**F2POS**, **IND**, **MARK**, **M2ARK**

COM Library Methods

GetIndexState, ResetIndexState

C Library Functions

acsc_GetIndexState, acsc_ResetIndexState

3.2.11 M2ARK

Description

M2ARK is a 64 member (one for each axis) real array.

M2ARK stores the position of the last encountered **MARK2** signal in **IST**(axis_index)**.#MARK2**.

Tag

84

Comments

After power-up **IST(axis_index).#MARK2** is reset and the value of **MARK2** is undefined. When the motor encounters a **MARK2** signal, the bit is raised and the current **FPOS** position is latched to **IST**(axis_index)**. #MARK2**.

If the motion continues and the motor encounters another **M2ARK** signal, the new value is ignored as long as **#MARK2** = 1.

To resume the latching logic, **IST**(axis_index)**. #MARK2** must be explicitly cleared with the command **IST**(axis_index)**. #MARK2**=0.

Accessibility

Read-Only

Related ACSPL+ Variables

IST, **FPOS**,**F2POS**, **MARK**

COM Library Methods

GetIndexState, ResetIndexState

C Library Functions

acsc_GetIndexState, acsc_ResetIndexState

3.2.12 MARK

Description

MARK is a 64 member (one for each axis) real array.

MARK stores the position of the last encountered **MARK1** signal in **IST**(axis_index)**.#MARK**.

Tag

85

Comments

After power-up, **IST(axis_index).#MARK** is reset and the value of **MARK** is undefined. When the motor encounters a **MARK1** signal, the bit is raised and the current **FPOS** position is latched to **IST(axis_index).#MARK**.

If the motion continues and the motor encounters another **MARK1** signal, the new value is ignored as long as **#MARK** = 1.

To resume the latching logic, **IST(axis_index).#MARK** must be explicitly cleared with the command **IST(axis_index).#MARK**=0.

Accessibility

Read-Only

Related ACSPL+ Variables

IST, **FPOS**, **F2POS**, **IND**, **M2ARK**, **IENA**

COM Library Methods

GetIndexState, ResetIndexState

C Library Functions

acsc_GetIndexState, acsc_ResetIndexState

3.2.13 MPOS

Description

MPOS is a 64 member (one for each axis) real array.

MPOS defines the current master position value for the axis in user units.

Tag

89

Comments

MASTER must precede **MPOS** for the specified axis. **MPOS** updates each controller cycle according to the formula specified in **MASTER**.

Accessibility

Read-Only

Related ACSPL+ Commands

MASTER, **SLAVE**

Related ACSPL+ Variables

FPOS, **F2POS**, **APOS**

COM Library Methods

SetMaster, Slave

C Library Functions

acsc_SetMaster, acsc_Slave

3.2.14 MST

Description

MST is a 64 member (one for each axis) real array containing a set of bits that display the current motor state, as given in **Table 23**:

Table 23 MST Bit Descriptions

Bit Name	No	Description
#ENABLED	0	0: motor is disabled 1: motor is enabled.
#OPEN	1	0: motor is operating with closed loop control 1: motor is operating with open loop control.
#INPOS	4	Obsolete bit. Use the #MOVE bit to get indication on the state of the motor.
#MOVE	5	0: Motor is not moving and has reached the target position (see variables **TARGRAD** and **SETTLE**) 1: Motor is moving
#ACC	6	0 = Motor is not accelerating 1 = Motor is accelerating.

Tag

90

Accessibility

Read-Only

Related ACSPL+ Commands

All motion commands.

Related ACSPL+ Variables

FPOS, **F2POS**, **APOS**, **RPOS**

COM Library Methods

GetMotorState

C Library Functions

acsc_GetMotorState

3.2.15 PE

Description

PE is a 64 member (one for each axis) real array.

PE displays the difference between **RPOS** and **FPOS** (the current position error) and denotes a noncritical position error.

Note	
	*The **PE** value is valid only if the motor is enabled.*

Tag

98

Accessibility

Read-Only

Related ACSPL+ Commands

All motion commands.

Related ACSPL+ Variables

FPOS, **RPOS**, **FAULT**

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.2.16 RACC

Description

RACC is a 64 member (one for each axis) real array.

RACC defines the current reference acceleration value for the motor in user-defined units.

Tag

106

Comments

RACC updates each controller cycle, and is calculated by digital differentiation of **RVEL**.

Accessibility

Read-Only

Related ACSPL+ Commands

All motion related commands.

Related ACSPL+ Variables

RVEL, **RPOS**

COM Library Methods

SetAcceleration, GetAcceleration

C Library Functions

SetAcceleration, GetAcceleration

3.2.17 ROFFS

Description

ROFFS is a 64 member (one for each axis) real array that stores the Reference Offset.

Tag

107

Comments

As long as the motor is in the default connection (**MFLAGS(***axis***).#DEFCON** = 1), offset **ROFFS** is zero. Once a user specifies connect formula such as:

CONNECT RPOS(0) = F(…)

the controller calculates offset **ROFFS(0)** to prevent a sudden change in **RPOS(0)** that may cause the motor to jump. The controller then calculates:

RPOS(0) = F(…) + ROFFS(0)

each controller cycle.

The controller recalculates **ROFFS** to prevent motor jump when the commands **CONNECT**, **SET**, **ENABLE/ENABLEALL**, **DISABLE**, **KILL** are executed. **ROFFS** reads the current value of the offset.

Note	*Watching the* **ROFFS** *value facilitates development and debugging of applications with complex kinematics.*

Accessibility

Read-Only

Related ACSPL+ Commands

CONNECT, **SET**, **ENABLE/ENABLEALL**, **DISABLE**, **KILL**

Related ACSPL+ Variables

MFLAGS(axis_index).#DEFCON (bit 17 = Default Connection)

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.2.18 RPOS

Description

RPOS is a 64 member (one for each axis) real array that stores the current desired motor reference position.

Tag

108

Comments

RPOS updates each MPU cycle according to the connection specified for the motor, see **CONNECT**.

When the motor is disabled, **RPOS** = **FPOS**.

Accessibility

Read-Only

Related ACSPL+ Commands

SET, **CONNECT**, and all motion commands.

Related ACSPL+ Variables

FPOS, **RVEL**, **RACC**

COM Library Methods

GetRPositon, SetRPosition

C Library Functions

acsc_GetRPositon, acsc_SetRPosition

3.2.19 RVEL

Description

RVEL is a 64 member (one for each axis) real array that stores the current motor reference velocity in user-defined units.

Tag

109

Accessibility

Read-Only

Related ACSPL+ Commands

All motion commands.

Related ACSPL+ Variables

RPOS, **RACC**, **FVEL**

COM Library Methods

GetRVelocity

C Library Functions

acsc_GetRVelocity

3.3 Data Collection Variables

The Data Collection variables are:

Name	Description
DCN	Axis Data Collection, Number of Samples
DCP	Axis Data Collection, Period
S_DCN	System Data Collection, Number of Samples
S_DCP	System Data Collection, Period
S_ST	State of System Data Collection

3.3.1 DCN

Description

DCN is a 64 member (one for each axis) integer array that stores the number of data collection samples per given axis.

Tag

19

Comments

DCN stores a defined number of axis data collection samples, as follows:

1. While an axis data collection is in progress **DCN** displays the index of the array element that stores the next sample.
2. When an axis data collection terminates for the corresponding axis, **DCN** stores the number of actually collected samples. If the data collection terminates automatically, the variable is always equal to the requested number of samples specified in **DC**. If **STOPDC** terminates data collection, **DCN** may contain less than the specified number of samples.

If an axis data collection is in progress, **DCN** increments each time the next sample is stored. When the data collection terminates, the **DCN** holds the last value, until the next data collection starts for the same axis.

Accessibility

Read-Only

Related ACSPL+ Commands

DC, **STOPDC**

Related ACSPL+ Variables

AST, **DCP**

COM Library Methods

DataCollection, StopCollect, WaitCollectEnd

C Library Functions

acsc_DataCollection, acsc_StopCollect, acsc_WaitCollectEnd

3.3.2 DCP

Description

DCP is a 64 member (one for each axis) real array that stores axis data collection samples based on a specified sampling *period*. When a axis data collection terminates, **DCP** stores the sampling period.

Tag

21

Comments

DCP is generally equal to the specified *period*, however because *period* is rounded to an integer number of controller cycles, the actual *period* may differ from the *period* specified in the **DC** command.

If **DC/t** (temporal data collection) was executed, **DCP** may be greater than the requested minimal period.

When a system data collection starts, **DCP** is assigned a real data collection period.

Accessibility

Read-Only

Related ACSPL+ Commands

DC, **STOPDC**

Related ACSPL+ Variables

AST, **DCN**

COM Library Methods

DataCollection, StopCollect, WaitCollectEnd

C Library Functions

acsc_DataCollection, acsc_StopCollect, acsc_WaitCollectEnd

3.3.3 S_DCN

Description

S_DCN is a scalar integer that stores a defined number of system data collection samples.

Tag

111

Comments

S_DCN stores a defined number of system data collection samples, as follows:

1. While a system data collection is in progress **S_DCN** displays the index of the array element that stores the *next* sample.
2. When a system data collection terminates, the variable stores the number of actually collected samples. If the data collection terminates automatically, the variable is always equal to the requested number of samples specified in the dc command. If the data collection terminates due to the **STOPDC** command, the variable may be less than the requested number of samples.

For *cyclic* data collection **S_DCN** displays the current number of collected samples and changes as follows:

1. At the start of data collection, **S_DCN** is assigned with zero.
2. With each sampling, **S_DCN** is incremented until it reaches the specified size of the sample array
3. **S_DCN** remains unchanged - the newest sample overwrites the oldest, so the total number of samples remains the same.

As long as cyclic data collection is in progress, the application cannot use the sample array. After the cyclic data collection finishes, the controller repacks the sample array so that the first element represents the oldest sample and the last element represents the most recent sample.

Accessibility

Read-Only

Related ACSPL+ Commands

DC, **STOPDC**

Related ACSPL+ Variables

S_ST, **S_DCP**

COM Library Methods

DataCollection, StopCollect, WaitCollectEnd

C Library Functions

acsc_DataCollection, acsc_StopCollect, acsc_WaitCollectEnd

3.3.4 S_DCP

Description

S_DCP is a scalar integer that stores the period of system data collection sampling.

Tag

112

Comments

When a system data collection terminates, the **S_DCP** stores the sampling *period*. Unless a temporal data collection was executed, the variable is always equal to the requested period specified in the **DC** command.

Note	
	S_DCP *is generally equal to the specified period, however because the period is rounded to an integer number of controller cycles, the actual period may differ from the period specified in the* ***DC*** *command.*

Accessibility

Read-Only

Related ACSPL+ Commands

DC, **STOPDC**

Related ACSPL+ Variables

S_ST, **S_DCN**

COM Library Methods

DataCollection, StopCollect, WaitCollectEnd

C Library Functions

acsc_DataCollection, acsc_StopCollect, acsc_WaitCollectEnd

3.3.5 S_ST

Description

S_ST is a scalar integer variable that provides the state of System Data Collection.

Tag

120

Comments

S_ST provides a bit that indicates if system data collection is currently in progress.

Bit 3:

0 - System data collection off

1 - System data in progress.

Accessibility

Read-Only

Related ACSPL+ Commands

DC, **STOPDC**

Related ACSPL+ Variables

S_DCN, **S_DCP**

COM Library Methods

DataCollection, StopCollect, WaitCollectEnd

C Library Functions

acsc_DataCollection, acsc_StopCollect

3.4 Input and Output Variables

The Input and Output variables are:

Name	Description
AIN	Analog Inputs
AOUT	Analog Outputs
COMMCH	Returns the last activated communication channel.
DCOM	Drive command - in open loop
EXTIN	Extended digital inputs (HSSI)
EXTOUT	Extended digital outputs (HSSI)
IN	General Purpose Digital Inputs
OUT	General Purpose Digital Outputs

3.4.1 AIN

Description

AIN is a 16 member (one for each analog input) integer array and is used for defining the level of an analog signal from an external source such as a sensor or a potentiometer.

Syntax

AIN(*index) = value*

Arguments

index	A number between 0 and 63.
value	**value** ranges from -32767 to +32767, Default = 0. These values, however, are SPiiPlus model-dependent - see below.

Tag

4

Comments

None

Accessibility

Read-Only

COM Library Methods

GetAnalogInput

C Library Functions

acsc_GetAnalogInput

3.4.2 AOUT

Description

AOUT is a 64 member (one for each analog output) integer array and is used for defining the level of a general purpose analog signal that is sent to an external device.

Syntax

AOUT(*index) = value*

Arguments

index	A number between 0 and 63.
value	**value** ranges from -32767 to +32767, Default = 0.

Tag

5

Comments

1. Some aspects of **AOUT** are model-dependent, including the number of analog outputs and type of analog inputs (differential or single-ended).
2. In SPiiPlus controllers (not CM) **AOUT** can be used only when the axis is defined as Dummy - see **MFLAGS**.
3. To define the analog output command to a drive (connected to a motor) in open loop, refer to **DCOM**.

Accessibility

Read-Write

Related ACSPL+ Variables

MFLAGS

COM Library Methods

GetAnalogOutput, SetAnalogOutput

C Library Functions

acsc_GetAnalogOutput, acsc_SetAnalogOutput

3.4.3 EXTIN

Description

EXTIN is a 64 member integer array which reads the current state of the HSSI inputs. Each element of the array reads 16 bits; therefore, up to 1024 inputs can be read. The number of inputs depends on the HSSI module.

For details about the HSSI, see the *HSSI Modules Hardware Guide*.

Tag

42

Accessibility

Read-Only

Related ACSPL+ Variables

EXTOUT, **IN**

COM Library Methods

GetExtInput, GetExtInputPort

C Library Functions

acsc_GetExtInput, acsc_GetExtInputPort

3.4.4 EXTOUT

Description

EXTIN is a 64 member integer array which can be used for reading or setting the current state of the HSSI outputs. Each element of the array reads 16 bits; therefore, up to 1024 outputs can be read or written. The number of outputs depends on the HSSI module.

For details about the HSSI, see the *HSSI Modules Hardware Guide*.

Syntax

EXTOUT(*index) = value*

Arguments

index	A number between 0 and 63.
value	**value** ranges from -2147483648, 2147483647, Default = 0.

Tag

43

Accessibility

Read-Write

Related ACSPL+ Variables

EXTIN, **OUT**

COM Library Methods

GetExtOutput, SetExtOutput, GetExtOutputPort, SetExtOutputPort

C Library Functions

acsc_GetExtOutput, acsc_SetExtOutput, acsc_GetExtOutputPort, acsc_SetExtOutputPort

3.4.5 IN

Description

IN is an integer array and stores the current state of the General Purpose digital inputs.

Syntax

IN(*port).bit*

Arguments

port	A number between 0 and 63.
bit	**bit** can be 0-31.

Tag

71

Comments

General Purpose inputs are represented by bits 0..31 of **IN**(port). Each bit reports the state of one General Purpose input.

For example, entering the query command **?IN(0).0** in the MMI **Communication Terminal** will return "0" if inputs #0 is non-active or "1" when active.

Note	*In some SPiiPlus controllers, the digital input pins can also be used as* **MARK**.

Accessibility

Read-Only

Related ACSPL+ Variables

OUT, **EXTOUT**

COM Library Methods

GetInput, GetInputPort

C Library Functions

acsc_GetInput, acsc_GetInputPort

3.4.6 OUT

Description

OUT is an integer arrau and can be used for reading or writing the current state of the General Purpose digital outputs.

Syntax

OUT(*port).bit*

Arguments

port	A number between 0 and 63.
bit	**bit** can be 0-31.

Tag

94

Comments

General purpose outputs are represented by bits 0..31 of **OUT**(port). Each bit reports the state of one general purpose output for the given port.

For example, the query command **?OUT(23).0** = 1 through the **Communication Terminal** will activate the outputs #0 of port 23.

Note	
	In some SPiiPlus controllers the digital output pins can also be used as **PEG** *- see* **ASSIGNPEG**.

Accessibility

Read-Write

Related ACSPL+ Variables

IN, **EXTIN**

COM Library Methods

SetOutput, GetOutput, SetOutputPort, GetOutputPort

C Library Functions

acsc_SetOutput, acsc_GetOutput, acsc_SetOutputPort, acsc_GetOutputPort

3.5 Monitoring Variables

The Monitoring variables are:

Name	Description
SMNA	Software Monitor Address
SMNV	Software Monitor Value
TIME	Elapsed Time
USAGE	MPU Usage

3.5.1 SMNA

Description

SMNA is a scalar integer and is used for defining the SP address of a variable to be monitored by **SMNV**.

Syntax

SMNA = *value*

Arguments

value	**value** ranges between -2147483648 to 2147483647, Default = 0.

Tag

125

Comments

After **SNMA** is set, the requested SP variable is copied to **SNMV** each MPU cycle. The **SNMV** variable can be used in ACSPL+ expressions, in immediate commands, or in data collection.

Accessibility

Read-Write

Related ACSPL+ Variables

SMNV

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.5.2 SMNV

Description

SMNV is a scalar integer used for storing a monitored value requested by **SMNA**.

Tag

126

Comments

After **SNMA** is set, the requested SP is copied to **SNMV** each MPU cycle. **SNMV** can be used in ACSPL+ expressions, in immediate commands or in data collection.

The values it can store range from -2147483648 to 2147483647, Default = 0.

Accessibility

Read-Only

Related ACSPL+ Variables

SMNA

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.5.3 TIME

Description

TIME is a scalar integer variable that provides the defines the elapsed time (in milliseconds) from the controller power-up.

Tag

134

Accessibility

Read-Only

3.5.4 USAGE

Description

USAGE is a scalar integer used for storing the amount of MPU usage as a percentage of the real-time tasks in the controller cycle during the execution of real-time tasks.

Tag

137

Comments

The real-time tasks always have the greatest priority. If the usage reaches 90% or more, the response time of the controller deteriorates, in addition, it is dangerous and may cause jerks in the motion profile.

The **USAGE** variable value can be used in autoroutines for halting the application should usage exceed a certain value.

Accessibility

Read-Only

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.6 Motion Variables

The Motion variables are:

Name	Description
ACC	Default Acceleration
DEC	Default Deceleration
GACC	Group Acceleration
GJERK	Group Jerk
GMOT	Motion Number
GMQU	Motion Queue
GMTYPE	Motion Type
GPATH	Group Path
GPHASE	Motion Phase
GRTIME	Remaining Motion Time
GSEG	Motion Segment

Name	Description
GSFREE	Free Motion Segments
GVEC	Group Vector
GVEL	Group Velocity
JERK	Default jerk
KDEC	Default kill deceleration
NVEL	Sets a non-zero axis velocity in stepper motor applications
TPOS	Target position for track motion
VEL	Velocity
VELBRK	Braking Velocity

3.6.1 ACC

Description

ACC is a 64 member (one for each axis) real array used for defining the motion profile acceleration.

Syntax

ACC*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.22507e-308 to 1.79769e+308, Default = 100000.

Tag

1

Comments

For single-axis motion, **ACC** defines the axis acceleration. If the axis is a leading axis in a group, **ACC** defines the vector acceleration of the common motion.

If **ACC** is changed when a motion is in progress, the change does not affect currently executing motions, or motions that were created before the change.

Accessibility

Read-Write

Related ACSPL+ Commands

IMM, **SLAVE**, and all motion commands where the profile is generated by the controller.

Related ACSPL+ Variables

DEC, **JERK**, **KDEC**, **VEL**

COM Library Methods

SetAcceleration. GetAcceleration, SetAccelerationImm

C Library Functions

acsc_SetAcceleration. acsc_GetAcceleration, acsc_SetAccelerationImm

3.6.2 DEC

Description

DEC is a 64 member (one for each axis) real array used for specifying the motion profile deceleration in milliseconds.

Syntax

DEC*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.22507e-308 to 1.79769e+308, Default = 100000.

Tag

22

Comments

For single-axis motion, **DEC** defines axis deceleration. If the axis is a leading axis in a group, **DEC** defines the vector deceleration of the common motion.

If **DEC** is changed when a motion is in progress, the change does not affect currently executing motions or motions that were created before the change.

Accessibility

Read-Write

Related ACSPL+ Commands

HALT, **IMM**, **SLAVE**

Related ACSPL+ Variables

ACC, **JERK**, **KDEC**, **VEL**

COM Library Methods

SetDeceleration. GetDeceleration, SetDecelerationImm

C Library Functions

acsc_SetDeceleration. acsc_GetDeceleration, acsc_SetDecelerationImm

3.6.3 GACC

Description

GACC is a 64 member (one for each axis) real array used for deriving the vector acceleration of a group motion.

For example when the three axes 0, 1 and 2 are moving as a group, the **GACC** is calculated by:

$$GACC = \sqrt{(Acceleration_0)^2 + (Acceleraton_1)^2 + (Acceleration_2)^2}$$

Note	
	***GPATH**, **GVEL**, **GACC**, **GJERK**, **GPHASE**, and **GRTIME** are not updated while the motion is in progress.*

Tag

55

Accessibility

Read-Only

Related ACSPL+ Commands

GROUP

Related ACSPL+ Variables

GVEL, **GJERK**, **GPATH**, **GPHASE**, **GRTIME**

COM Library Methods

SetAcceleration. GetAcceleration, SetAccelerationImm

C Library Functions

acsc_SetAcceleration. acsc_GetAcceleration, acsc_SetAccelerationImm

3.6.4 GJERK

Description

GJERK is a 64 member (one for each axis) real array used for deriving the vector acceleration of a group motion.

For example when the three axes 0, 1 and 2 are moving as a group, the **GJERK** is calculated by:

$$GJERK = \sqrt{(Jerk_0)^2 + (Jerk_1)^2 + (Jerk_2)^2}$$

Note	
	GPATH, **GVEL**, **GACC**, **GJERK**, **GPHASE**, *and* **GRTIME** *are not updated while the motion is in progress.*

Tag

57

Accessibility

Read-Only

COM Library Methods

GetJerk, SetJerk, SetJerkImm

C Library Functions

acsc_GetJerk, acsc_SetJerk, acsc_SetJerkImm

3.6.5 GMOT

Description

GMOT is a 64 member (one for each axis) real array that defines the ordinal number of the current motion.

Tag

58

Comments

The **GMOT** value is valid only if one of the following is true:

- Single-axis motion in progress
- The axis is a leading axis in a group and motion in the group is in progress

After power-up, **GMOT** is zero and increments each time a motion of the corresponding axis/axis group terminates.

GMOT resets to zero each time the axis group is created or split.

Accessibility

Read-Only.

Related ACSPL+ Commands

GROUP, **SPLIT**

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.6.6 GMQU

Description

GMQU is a 64 member (one for each axis) integer array that defines the total number of motions in the motion queue including the currently executing motion. The maximum motion queue per axis is 5.

Tag

59

Comments

GMQU is valid only if one of the following is true:

- Single-axis motion in progress
- The axis is a leading axis in a group and motion in the group is in progress

After power-up **GMQU** is zero. The variable is incremented by one each time a new motion of the corresponding axis/axis group is issued. It is decremented by one each time a motion of the corresponding axis/axis group terminates.

GMQU resets to zero each time an axis is regrouped, i.e., a group that contains the axis is created or split-up.

Accessibility

Read-Only

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.6.7 GMTYPE

Description

GMTYPE is a 64 member (one for each axis) integer array. The MPU updates **GMTYPE** each time a motion involving the corresponding axis or axis group starts or terminates.

Tag

60

Comments

GMTYPE is updated according to the type of the motion as follows:

0 - no motion

1 - **PTP** motion

2 - **MPTP...ENDS** motion

3 - **TRACK** motion

4 - **MSEG...ENDS** motion

5 - **JOG** motion

6 - **SLAVE** motion

7 - **PATH...ENDS** motion

8 - **PVSPLINE...ENDS** motion

Accessibility

Read-Only

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.6.8 GPATH

Description

GPATH is a 64 member (one for each axis) real array. **GPATH** defines the current path value, defined as the distance from the motion origin to the current motion point.

Tag

61

Comments

GPATH updates each MPU cycle if one of the following is true:

- Single-axis motion in progress
- The axis is a leading axis in a group and motion in the group is in progress

If either of these conditions is not true, **GPATH** retains its previous value.

For single-axis motion, **GPATH** defines a positive distance from the initial point of the motion.

If the axis is a leading axis, **GPATH** defines a vector distance along the trajectory from the motion origin.

Note	**GPATH**, **GVEL**, **GACC**, **GJERK**, **GPHASE**, *and* **GRTIME** *are not updated while the motion is in progress.*

Accessibility

Read-Only

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.6.9 GPHASE

Description

GPHASE is a 64 member (one for each axis) real array. **GPHASE** defines the current phase of a motion.

Tag

62

Comments

GPHASE can have the following values:

0 - no motion

1 - acceleration buildup

2 - constant acceleration

3 - acceleration finishing

4 - constant velocity

5 - deceleration buildup

6 - constant deceleration

7 - deceleration finishing

8 - kill deceleration

The following values apply only in the case of **MASTER-SLAVE** motion:

9 - asynchronous phase of master-slave motion

10 - synchronous phase of master-slave motion

11 - stalled phase of master-slave motion.

12 - dwell phase in **JOG** or **MPTP...ENDS** motions, or no defined target point in **PATH...ENDS**, **PVSPLINE...ENDS** or **MPTP...ENDS** motions.

Note	
	GPATH, **GVEL**, **GACC**, **GJERK**, **GPHASE**, *and* **GRTIME** *are not updated while the motion is in progress.*

Accessibility

Read-Only

COM Library Methods

GetCallbackMask

C Library Functions

acsc_GetCallbackMask

3.6.10 GRTIME

Description

GRTIME is a 64 member (one for each axis) real array. **GRTIME** defines an estimated value of time (in milliseconds) remaining until the end of the current motion.

Tag

63

Comments

GRTIME updates each MPU cycle if one of the following is true:

- Single-axis motion in progress
- The axis is a leading axis in a group and motion in the group is in progress

GRTIME does not update if the motion is **JOG** or **MASTER-SLAVE**. If **GRTIME** does not update, it retains its previous value.

Normally, 1-2 msec after motion starts, **GRTIME** accepts the correct value. In rare cases, the **GRTIME** value remains high during motion phases 1 and 2, and accepts correct value at the beginning of phase 3.

Note	
	GPATH, **GVEL**, **GACC**, **GJERK**, **GPHASE**, *and* **GRTIME** *are not updated while the motion is in progress.*

Accessibility

Read-Only

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.6.11 GSEG

Description

GSEG is a 64 member (one for each axis) real array. **GSEG** defines the ordinal number of the currently executing segment.

Tag

64

Comments

GSEG updates only under one of the following conditions

- Single-axis motion in progress, or
- The axis is a leading axis in a group and motion in the group is in progress

If either of these conditions is not true, **GSEG** retains its previous value.

GSEG updates as follows:

- If the current motion in the axis/axis group is not **MSEG...ENDS**, the **GSEG** value is -1.
- The value resets to zero when a multi segment motion starts
- The value increments each time when the motion passes from one segment to the next.
- The value decrements each time the motion passes from one segment to the previous (possible only in master-slave motion).
- Because the motion returns to the start point in cyclic motion, **GSEG** may appear greater than the number of a segment in the motion, if the motion overruns the segment sequence in positive direction.
- For master-slave cyclic motion, **GSEG** may appear negative, if the motion overruns the segment sequence in a negative direction.

Accessibility

Read-Only

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.6.12 GSFREE

Description

GSFREE is a 64 member (one for each axis) integer array. **GSFREE** is updated for the leading axis with the number of free cells in the segment queue.

Tag

65

Comments

If **GSFREE** is zero, the segment queue is full and the next coming **POINT** or **MPOINT** command will be delayed until the required number of cells are freed.

Accessibility

Read-Only

Related Standard Variable

GSEG

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.6.13 GVEC

Description

GVEC is a 64 member (one for each axis) integer array. **GVEC** is updated each MPU cycle, if a motion involving the axis is in progress. If the motion is not in progress, **GVEC** retains its previous value.

Tag

66

Comments

In single-axis motion, **GVEC** = 1 or -1, depending on the motion direction.

In multi-axis group motion, **GVEC** values for all axes in the group are updated each MPU cycle and together build up a tangent vector for the motion trajectory.

GVEC can also be used for retrieving a tangent vector.

Accessibility

Read-Only

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.6.14 GVEL

Description

GVEL is a 64 member (one for each axis) real array used for deriving the vector velocity of a group motion.

For example when the three axes 0, 1 and 2 are moving as a group, the **GVEL** is calculated by:

$$GVEL = \sqrt{(Velocity_0)^2 + (Velocity_1)^2 + (Velocity_2)^2}$$

Note	
	GPATH, **GVEL**, **GACC**, **GJERK**, **GPHASE**, *and* **GRTIME** *are not updated while the motion is in progress.*

Tag

67

Accessibility

Read-Only

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.6.15 JERK

Description

JERK is a 64 member (one for each axis) real array used for defining the jerk of the motion profile.

Syntax

JERK*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.22507e-308 to 1.79769e+308, Default = 2e+007.

Tag

80

Comments

For single-axis motion, **JERK** defines the axis jerk. If the axis is a leading axis in a group, **JERK** defines vector jerk of the common motion.

If **JERK** is changed when a motion is in progress, the change does not affect currently executing motions, or motions that were created before the change.

Accessibility

Read-Write

Related ACSPL+ Commands

IMM, and all motion commands where the profile is generated by the controller.

Related ACSPL+ Variables

ACC, **DEC**, **KDEC**, **VEL**

COM Library Methods

GetJerk, SetJerk, SetJerkImm

C Library Functions

acsc_GetJerk, acsc_SetJerk, acsc_SetJerkImm

3.6.16 KDEC

Description

KDEC is a 64 member (one for each axis) real array used for defining deceleration when a motion is killed by the user or fails due to a fault.

Syntax

KDEC*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2.22507e-308 to 1.79769e+308, Default = 100000.

Tag

81

Comments

For single-axis motion, the value defines axis deceleration. If the axis is a leading axis in a group, **KDEC** defines the vector deceleration when the common motion is killed or fails.

If **KDEC** is changed when a motion is in progress, the change does not affect currently executing or motions that were created before the change.

Accessibility

Read-Write

Related ACSPL+ Commands

None

Related ACSPL+ Variables

ACC, **DEC**, **JERK**, **VEL**

COM Library Methods

SetKillDeceleration, GetKillDeceleration, SetKillDecelerationImm

C Library Functions

acsc_SetKillDeceleration, acsc_GetKillDeceleration, acsc_SetKillDecelerationImm

3.6.17 NVEL

Description

NVEL is a 64 member (one for each axis) real array used for specifying the start velocity for an axis in stepper motor applications.

Syntax

NVEL*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 0 to 1.79769e+308, Default = 0.

Tag

91

Comments

1. An **NVEL** element affects the motion of the corresponding axis and the multi-axis motions if the axis is a leading axis in the group.
2. If an element is zero, the normal motion profile starts from zero velocity and finishes at zero velocity.

   If an element is non-zero, at the beginning of motion the velocity immediately jumps to the **NVEL** value and then continues the regular motion profile. At the end of the motion, the motion approaches the final point at the velocity specified by **NVEL**, and then immediately drops to zero. For example, **KILL** and **HALT** slow the velocity to the value specified in **NVEL**, and then the velocity drops to zero.

Related ACSPL+ Commands

IMM, and all motion commands where the profile is generated by the controller.

Related ACSPL+ Variables

VEL

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.6.18 TPOS

Description

TPOS is a 64 member (one for each axis) real array used for defining or updating the target position in TRACK motion.

Syntax

TPOS(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from -1.79769e+308 to 1.79769e+308, Default = 0.

Tag

135

Comments

The controller update occurs as follows:

- When the controller executes **PTP** motion, the axes' target coordinates are stored in the **TPOS** elements.
- During **MPTP...ENDS** motion, the controller updates the target coordinates each time motion to the next point starts.
- When the controller executes **TRACK** motion, the axes' target coordinates are stored in the **TPOS** elements.

Accessibility

Read-Write

Related ACSPL+ Commands

TRACK

COM Library Methods

Track

C Library Functions

acscTrack

3.6.19 VEL

Description

VEL is a 64 member (one for each axis) real array used for defining the default velocity of the motion profile. If a motion command does not specify a specific velocity, the default value is used.

Syntax

VEL(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from -1.79769e+308 to 1.79769e+308, Default = 10000.

Tag

139

Comments

For single-axis motion, the value defines axis velocity. If the axis is a leading axis in a group, its value defines a vector velocity of common motion.

If **VEL** is changed when a motion is in progress, the change does not affect currently executing motions or motions that were created before the change.

Accessibility

Read-Write

Related ACSPL+ Commands

IMM, and all motion commands where the profile is generated by the controller.

Related ACSPL+ Variables

ACC, **DEC**, **JERK**, **KDEC**,

COM Library Methods

SetVelocity, GetVelocity, SetVelocityImm

C Library Functions

acsc_SetVelocity, acsc_GetVelocity, acsc_SetVelocityImm

3.6.20 VELBRK

Description

VELBRK is a 64 member (one for each axis) real array used for defining dynamic braking.

Syntax

VELBRK(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 0 to 1.79769e+308, Default = 0.

Tag

140

Comments

If **MFLAGS(axis_index).#BRAKE** is = 1, the controller will apply dynamic braking to stop the motor when the axis is disabled and the feedback velocity is less than the predefined value of **VELBRK**.

Accessibility

Read-Write

Related ACSPL+ Variables

MFLAGS(axis_index).#BRAKE

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.7 Program Execution Control Variables

The Program Execution Control variables are:

Name	Description
ONRATE	Autoroutine Rate
PCHARS	Program Size in Characters
PERL	Program Error Line
PERR	Program Error
PEXL	Executed Line
PFLAGS	Program Flags
PLINES	Number of Lines
PRATE	Program Rate
PST	Program State

3.7.1 ONRATE

Description

ONRATE is a 16 element (one for each program buffer plus one for the D-Buffer) integer array used for controlling the autoroutine execution rate.

Syntax

ONRATE*(buffer_index) = value*

Arguments

buffer_index	**buffer index** - a number between 0 and 16 (16 being the D-Buffer).
value	**value** ranges from 1 to 10, Default = 1.

Tag

93

Comments

ONRATE is set through **SPiiPlus MMI Application Studio → Toolbox → Application Development → Program Manager → Program Buffer Parameters**.

When an autoroutine executes in the program buffer, the execution rate is **ONRATE** lines per each MPU cycle. The normal rate of program execution (when no autoroutine is activated) is defined by **PRATE**.

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.7.2 PCHARS

Description

PCHARS is a 16 element (one for each program buffer plus one for the D-Buffer) integer array that stores the total number of characters stored in the buffer.

Tag

95

Accessibility

Read-Only

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVarible

3.7.3 PERL

Description

PERL is an 16 element (one for each program buffer plus one for the D-Buffer) integer array that stores the line number where the error occurred.

Tag

99

Comments

If an error occurs during ACSPL+ program execution, the controller stores the line number where the error occurred in the corresponding element of the **PERL** array.

Accessibility

Read-Only

Related ACSPL+ Variables

PERR

COM Library Methods

GetProgramError

C Library Functions

acsc_GetProgramError

3.7.4 PERR

Description

PERR is an 16 element (one for each program buffer plus one for the D-Buffer) integer array that stores an error code.

Tag

100

Comments

If an error occurs during ACSPL+ program execution, the controller stores the error code in the corresponding element of the **PERR** array.

Accessibility

Read-Only

Related ACSPL+ Variables

PERL

COM Library Methods

GetProgramError

C Library Functions

acsc_GetProgramError

3.7.5 PEXL

Description

PEXL is an 16 element (one for each program buffer plus one for the D-Buffer) integer array that stores the number of the currently executed line.

Tag

101

Comments

PEXL stores the number of the currently executed line in the buffer. If the program has not executed, the variable reads zero.

Accessibility

Read-Only

Related ACSPL+ Variables

PERL, **PERR**

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.7.6 PFLAGS

Description

PFLAGS is an 16 element (one for each program buffer plus one for the D-Buffer) integer array containing a set of bits that defines the behavior of the program buffer.

Syntax

PFLAGS*(buffer_index).(bit) = 0|1*

Arguments

buffer_index	**buffer index** - a number between 0 and 16 (16 being the D-Buffer).
bit	See **Table 24**

Table 24 PFLAGS Bit Description

Bit Name	No.	Description
#NOAUTO	0	0 (default): Autoroutines (if exist in the buffer) are enabled. 1: Autoroutines (if exist in the buffer) are disabled.
#NOEDIT	1	0 (default): User program can be viewed and edited. 1: User program can be viewed but cannot be edited, see **PROTECT** Terminal command.
#DYNAMIC	2	0 (default): Buffer works in normal order. 1: Not applicable (obsolete option)
#PRIVLG	4	0 (default): Buffer works in normal order. 1: Sets the buffer as privileged which means that the program in the buffer can change the values of protected variables, start and stop other ACSPL+ programs, and execute any other action that in a regular buffer would cause a protection violation.
#DEBUG	5	0 (default): Buffer works in normal order. 1: Not applicable (obsolete option)
#NOVIEW	6	0 (default): The program is visible in the buffer. 1: The program in the buffer hidden from being viewed.

Tag

102

Comments

Note	
	PFLAGS *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Commands

PROTECT

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.7.7 PLINES

Description

PLINES is a 16 element (one for each program buffer plus one for the D-Buffer) integer array that stores the total number of lines stored in a buffer.

Tag

103

Accessibility

Read-Only

Related ACSPL+ Variables

PCHARS

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.7.8 PRATE

Description

PRATE is a 16 element (one for each program buffer plus one for the D-Buffer) integer array used for defining the program execution rate.

Syntax

PRATE*(buffer_index) = value*

Arguments

buffer_index	**buffer index** - a number between 0 and 16 (16 being the D-Buffer).
value	**value** ranges from 1 to 10, Default = 1.

Tag

104

Comments

PRATE is set through **SPiiPlus MMI Application Studio → Toolbox → Application Development → Program Manager → Program Buffer Parameters**.

PRATE defines the program execution rate. The execution rate is **PRATE** lines per each MPU cycle.

PRATE is used only if no autoroutine is activated in the buffer. While an autoroutine is executed, **ONRATE** defines execution rate.

For example, if the controller is configured so that **PRATE(2)** is 1, but **ONRATE(2)** is 4, the program in Buffer 2 will be executed one line per one controller cycle, and any autoroutine specified in Buffer 2 that interrupts the program will be executed four lines per one controller cycle. When the **RET** command that terminates the autoroutine is executed, the controller switches back to the rate of one line per one cycle.

Related ACSPL+ Variables

ONRATE

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.7.9 PST

Description

PST is a 16 element (one for each program buffer plus one for the D-Buffer) integer array containing a set of bits that display the current state of the program buffer.

The **PST** bits are detailed in **Table 25.**

Table 25 PST Bit Description

Bit Name	No.	Description
#COMPILED	0	0: Program in buffer is not compiled 1: Program in buffer is compiled
#RUN	1	0: Program is not running. 1: Program is running.
#SUSPEND	2	0: Program in buffer is not suspended. 1: Program is suspended after **STEP** or due to a breakpoint in debug mode.

Table 25 PST Bit Description

Bit Name	No.	Description
#DEBUG	5	0: Buffer works in normal order (default). 1: Not applicable (obsolete option)
#AUTO	7	0: Autoroutine is not running 1: Autoroutine is running.

Tag

105

Accessibility

Read-Only

COM Library Methods

GetProgramState

C Library Functions

acsc_GetProgramState

3.8 Safety Control Variables

The Safety Control variables are:

Name	Description
AERR	Axis Error
FAULT	Faults
FDEF	Default Response Mask
FMASK	Fault Mask
MERR	Motor Error
SAFIN	Safety Inputs
SAFINI	Safety Inputs Inversion
S_ERR	System Error
S_FAULT	System Faults
S_FDEF	System Default Response Mask
S_FMASK	System Fault Mask
S_SAFIN	System Safety Inputs
S_SAFINI	System Safety Inputs Inversion

3.8.1 AERR

Description

AERR is a 64 member (one for each axis) integer array that stores the termination codes.

Tag

2

Comments

When a motion starts, the controller zeroes the **AERR** elements for each axis involved in the motion. When the motion terminates for any reason, the controller stores the termination code in the corresponding **AERR** element. The element remains unchanged until the next motion starts for the corresponding axis.

Single-axis motion stores its termination code in the **AERR** element that corresponds to the axis. Multi-axis motion stores its termination code in the **AERR** element that corresponds to the leading axis of the motion.

Accessibility

Read-Only

Related ACSPL+ Variables

MERR

COM Library Methods

GetAxisState

C Library Functions

acsc_GetAxisState

3.8.2 FAULT

Description

FAULT is a 64 member (one for each axis) integer array with a set of bits that stores axis-related fault bits

The fault bits are detailed in **Table 26**.

Table 26 Axis Fault Bits (page 1 of 3)

Bit	Fault	Fault Description
0	#RL	RIGHT LIMIT. 1 = Right limit switch is activated.
1	#LL	LEFT LIMIT. 1 = Left limit switch is activated.
2	#NT	NETWORK ERROR. 1 = EtherCAT network error is activated.
3	#FAN	COOLING FAN ERROR. 1 = Cooling fan not operating detected.

Table 26 Axis Fault Bits (page 2 of 3)

Bit	Fault	Fault Description
4	#HOT	MOTOR OVERHEAT. 1 = Motor's temperature sensor indicates overheat.
5	#SRL	SOFTWARE RIGHT LIMIT. 1 = Axis reference position (**RPOS**) is greater than the software right limit margin (SRLIMIT).
6	#SLL	SOFTWARE LEFT LIMIT. 1 = Axis reference position (**RPOS**) is less than the software left limit margin (SLLIMIT).
7	#ENCNC	ENCODER NOT CONNECTED. 1 = Primary encoder (for digital encoder type only) is not connected.
8	#ENC2NC	ENCODER 2 NOT CONNECTED. 1 = Secondary encoder (for digital encoder type only) is not connected.
9	#DRIVE	DRIVE ALARM. 1 = Signal from the drive reports a failure.
10	#ENC	ENCODER ERROR. 1 = Primary encoder miscounts.
11	#ENC2	ENCODER 2 ERROR. 1 = Secondary encoder miscounts.
12	#PE	POSITION ERROR. 1 = Position error (PE) has occurred. PE is defined by the following variables: ❑ **ERRI** - Maximum position error while the axis is idle ❑ **ERRV** - Maximum position error while the axis is moving with constant velocity ❑ **ERRA** - Maximum position error while the axis is accelerating or decelerating ❑ **DELI** - Delay on transition from **ERRA** to **ERRI** ❑ **DELV** - Delay on transition from **ERRA** to **ERRV**
13	#CPE	CRITICAL POSITION ERROR. 1 = Position error (#PE) exceeds the value of the critical limit. #CPE errors occur outside normal range of operation and #CPE > #PE. The critical limit depends on the axis state and is defined by the following variables: ❑ **CERRI** if the axis is idle (not moving) ❑ **CERRV** if the axis is moving with constant velocity ❑ **CERRA** if the axis is accelerating or decelerating ❑ **DELI** - Delay on transition from **ERRA** to **ERRI** ❑ **DELV** - Delay on transition from **ERRA** to **ERRV**
14	#VL	VELOCITY LIMIT. 1 = Absolute value of the reference velocity (**RVEL**) exceeds the limit defined by the **XVEL** parameter.

Table 26 Axis Fault Bits (page 3 of 3)

Bit	Fault	Fault Description
15	#AL	ACCELERATION LIMIT. 1 = Absolute value of the reference acceleration (**RACC**) exceeds the limit defined by the **XACC** parameter.
16	#CL	CURRENT LIMIT. 1 = **RMS** current calculated in the Servo Processor exceeds the limit value defined by the **XRMS** parameter.
17	#SP	SERVO PROCESSOR ALARM. 1 = Axis Servo Processor loses its synchronization with the MPU. The fault indicates a fatal problem in the controller.
20	#HSSINC	HSSI NOT CONNECTED. 1 = HSSI module is not connected.

Tag

47

Comments

FAULT indicates axis related fault bits as detected by the safety mechanism. When each of the faults is active (such as Left Limit), the corresponding fault bit becomes = 1 while the fault is active, and automatically reverts to 0 when the fault is no longer active.

- Each fault can be masked by **FMASK**.
- The logic of some faults can be inverted by **SAFINI**.
- The default response of each fault can be disabled by **FDEF**. In this case, any customized default response can be implemented by autoroutines - see **ON...RET**.

Note	
	FAULT *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

For a list of **S_FAULT** related system fault bits see **Table 31**.

Accessibility

Read-Only

Related ACSPL+ Variables

S_FAULT, **FDEF**, **S_FDEF**, **FMASK**, **S_FMASK**, **SAFIN**, **S_SAFIN**, **SAFINI**, **S_SAFINI**

COM Library Methods

GetFault

C Library Functions

acsc_GetFault

3.8.3 FDEF

Description

FDEF is a 64 member (one for each axis) integer array with a set of bits used for setting a default response to an axis fault.

Syntax

FDEF(*axis_index)[.bit_designator] = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
bit_designator	The **FDEF** bit designators are given in **Table 27.**
value	**value** ranges from -2147483648 to 2147483647, Default = -1.

Table 27 FDEF Bit Description (page 1 of 3)

Bit	Fault	Fault Description	Default Response (FDEF)
0	#RL	RIGHT LIMIT. 1 = Right limit switch is activated.	The controller kills the violating axis. As long as the fault is active, the controller kills any motion that tries to move the axis in the direction of the limit; however, motion within the permissible range is allowed.
1	#LL	LEFT LIMIT. 1 = Left limit switch is activated.	Same as for #RL.
2	#NT	NETWORK ERROR. 1 = EtherCAT network error detected.	Halts all program buffers and waits for receipt of network Sync signal.
3	#FAN	COOLING FAN ERROR. 1 = Cooling fan is non-operational.	No default response.
4	#HOT	MOTOR OVERHEAT. 1 = Motor's temperature sensor indicates overheat.	None.
5	#SRL	SOFTWARE RIGHT LIMIT. 1 = Axis reference position (**RPOS**) is greater than the software right limit margin (SRLIMIT).	The controller kills the violating axis. As long as the fault is active, the controller kills any motion that tries to move the axis in the direction of the limit. Motion in the direction out of the limit is allowed.
6	#SLL	SOFTWARE LEFT LIMIT. 1 = Axis reference position (**RPOS**) is less than the software left limit margin (SLLIMIT).	Same as #SRL.

Table 27 FDEF Bit Description (page 2 of 3)

Bit	Fault	Fault Description	Default Response (FDEF)
7	#ENCNC	ENCODER NOT CONNECTED. 1 = Primary encoder (for digital encoder type only) is not connected.	The controller disables the violating axis.
8	#ENC2NC	ENCODER 2 NOT CONNECTED. 1 = Secondary encoder (for digital encoder type only) is not connected.	No default response.
9	#DRIVE	DRIVE ALARM. 1 = Signal from the drive reports a failure.	The controller disables the violating axis. This fault is only detected when the axis is enabled. To catch this fault in an ACSPL+ program, write an autoroutine.
10	#ENC	ENCODER ERROR. 1 = Primary encoder miscounts.	The controller disables the violating axis. The faults remain active until the user resolves the problems and enables the axis again or executes **FCLEAR**.
11	#ENC2	ENCODER 2 ERROR. 1 = Secondary encoder miscounts.	Same as #ENC.
12	#PE	POSITION ERROR. 1 = Position error (PE) has occurred. PE is defined by the following variables: ❑ **ERRI** - Maximum position error while the axis is idle ❑ **ERRV** - Maximum position error while the axis is moving with constant velocity ❑ **ERRA** - Maximum position error while the axis is accelerating or decelerating ❑ **DELI** - Delay on transition from **ERRA** to **ERRI** ❑ **DELV** - Delay on transition from **ERRA** to **ERRV**	None.

Table 27 FDEF Bit Description (page 3 of 3)

Bit	Fault	Fault Description	Default Response (FDEF)
13	#CPE	CRITICAL POSITION ERROR. 1 = Position error (#PE) exceeds the value of the critical limit. #CPE errors occur outside normal range of operation and #CPE > #PE. The critical limit depends on the axis state and is defined by the following variables: ❑ **CERRI** if the axis is idle (not moving) ❑ **CERRV** if the axis is moving with constant velocity ❑ **CERRA** if the axis is accelerating or decelerating ❑ **DELI** - Delay on transition from **ERRA** to **ERRI** ❑ **DELV** - Delay on transition from **ERRA** to **ERRV**	The controller disables the violating axis.
14	#VL	VELOCITY LIMIT. 1 = Absolute value of the reference velocity (**RVEL**) exceeds the limit defined by the **XVEL** parameter.	The controller kills the violating axis.
15	#AL	ACCELERATION LIMIT. 1 = Absolute value of the reference acceleration (**RACC**) exceeds the limit defined by the **XACC** parameter.	The controller kills the violating axis.
16	#CL	CURRENT LIMIT. 1 = **RMS** current calculated in the Servo Processor exceeds the limit value defined by the **XRMS** parameter.	The controller disables the violating axis.
17	#SP	SERVO PROCESSOR ALARM. 1 = Axis Servo Processor loses its synchronization with the MPU. The fault indicates a fatal problem in the controller.	The controller disables the violating axis and kills the motion that involves the axis.
20	#HSSINC	HSSI NOT CONNECTED. 1 = HSSI module is not connected.	None.

Tag

48

Comments

When an **FDEF** bit = 1, the controller executes the default response when the corresponding fault occurs. If the **FDEF** bit = 0, the default response is disabled.

Not every fault has a default response. For a fault that has no default response, the corresponding **FDEF** bit is inoperative.

Note	**FDEF** *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

FAULT, **S_FAULT**, **S_FDEF**, **FMASK**, **S_FMASK**, **SAFIN**, **S_SAFIN**, **SAFINI**, **S_SAFINI**

COM Library Methods

GetResponseMask, SetResponseMask, GetFaultMask, SetFaultMask

C Library Functions

acsc_GetResponseMask, acsc_SetResponseMask, acsc_GetFaultMask, acsc_SetFaultMask

3.8.4 FMASK

Description

FMASK is a 64 member (one for each axis) integer array with a set of bits used for enabling or disabling each axis fault bit.

Syntax

FMASK(*axis_index)[.bit_designator] = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
bit_designator	The **FDEF** bit designators are given in **Table 28.**
value	**value** ranges from -2147483648 to 2147483647, Default=1040414435.

Table 28 FMASK Bit Description (page 1 of 2)

Bit	Fault	Fault Description
0	#RL	RIGHT LIMIT. 1 = Right limit switch is activated.
1	#LL	LEFT LIMIT. 1 = Left limit switch is activated.
2	#NT	NETWORK ERROR. 1 = EtherCAT network error detected.
3	#FAN	COOLING FAN ERROR. 1 = Cooling fan error is activated.
4	#HOT	MOTOR OVERHEAT. 1 = Motor's temperature sensor indicates overheat.
5	#SRL	SOFTWARE RIGHT LIMIT. 1 = Axis reference position (**RPOS**) is greater than the software right limit margin (SRLIMIT).
6	#SLL	SOFTWARE LEFT LIMIT. 1 = Axis reference position (**RPOS**) is less than the software left limit margin (SLLIMIT).
7	#ENCNC	ENCODER NOT CONNECTED. 1 = Primary encoder (for digital encoder type only) is not connected.
8	#ENC2NC	ENCODER 2 NOT CONNECTED. 1 = Secondary encoder (for digital encoder type only) is not connected.
9	#DRIVE	DRIVE ALARM. 1 = Signal from the drive reports a failure.
10	#ENC	ENCODER ERROR. 1 = Primary encoder miscounts.
11	#ENC2	ENCODER 2 ERROR. 1 = Secondary encoder miscounts.
12	#PE	POSITION ERROR. 1 = Position error (PE) has occurred. PE is defined by the following variables: ❑ **ERRI** - Maximum position error while the axis is idle ❑ **ERRV** - Maximum position error while the axis is moving with constant velocity ❑ **ERRA** - Maximum position error while the axis is accelerating or decelerating ❑ **DELI** - Delay on transition from **ERRA** to **ERRI** ❑ **DELV** - Delay on transition from **ERRA** to **ERRV**

Table 28 FMASK Bit Description (page 2 of 2)

Bit	Fault	Fault Description
13	#CPE	CRITICAL POSITION ERROR. 1 = Position error (#PE) exceeds the value of the critical limit. #CPE errors occur outside normal range of operation and #CPE > #PE. The critical limit depends on the axis state and is defined by the following variables: ❑ **CERRI** if the axis is idle (not moving) ❑ **CERRV** if the axis is moving with constant velocity ❑ **CERRA** if the axis is accelerating or decelerating ❑ **DELI** - Delay on transition from **ERRA** to **ERRI** ❑ **DELV** - Delay on transition from **ERRA** to **ERRV**
14	#VL	VELOCITY LIMIT. 1 = Absolute value of the reference velocity (**RVEL**) exceeds the limit defined by the **XVEL** parameter.
15	#AL	ACCELERATION LIMIT. 1 = Absolute value of the reference acceleration (**RACC**) exceeds the limit defined by the **XACC** parameter.
16	#CL	CURRENT LIMIT. 1 = **RMS** current calculated in the Servo Processor exceeds the limit value defined by the **XRMS** parameter.
17	#SP	SERVO PROCESSOR ALARM. 1 = Axis Servo Processor loses its synchronization with the MPU. The fault indicates a fatal problem in the controller.
20	#HSSINC	HSSI NOT CONNECTED. 1 = HSSI module is not connected.

Tag

51

Comments

The default value = 1 and causes the controller to check for the fault associated with that bit, as follows:

0 = the corresponding **FAULT** bit is disabled.

1 = the corresponding **FAULT** is enabled and examined each MPU cycle.

Note	
	FMASK *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

FAULT, **S_FAULT**, **S_FDEF**, **S_FMASK**, **SAFIN**, **S_SAFIN**, **SAFINI**, **S_SAFINI**

COM Library Methods

GetResponseMask, SetResponseMask, GetFaultMask, SetFaultMask

C Library Functions

acsc_GetResponseMask, acsc_SetResponseMask, acsc_GetFaultMask, acsc_SetFaultMask

3.8.5 MERR

Description

MERR is an 64 member (one for each axis) integer array that stores a code indicating the termination cause of the last motion of an axis.

The **MERR** return values are listed in **Table 29.**

Table 29 MERR Return Values (page 1 of 2)

Error Code	Description
0	Motor Enabled, in motion, or motion terminated by the user using **KILL**, **HALT**, or **DISABLE**.
5003	Motion was terminated by user
5007	Motor was disabled due to a generalized fault
5008	Motor was killed due to a generalized fault
5010	Motor failed: Right Limit
5011	Motor failed: Left Limit
5014	Motor failed: Overheat
5015	Motor failed: Software Right Limit
5016	Motor failed: Software Left Limit
5017	Motor failed: Encoder Not Connected
5018	Motor failed: Encoder 2 Not Connected
5019	Motor Failed: Drive Alarm
5020	Motor Failed: Encoder Alarm
5021	Motor Failed: Encoder 2 Alarm
5022	Motor Failed: Position Error
5023	Motor Failed: Critical Position Error
5024	Motor Failed: Velocity Limit
5025	Motor Failed: Acceleration Limit
5026	Motor Failed: Overcurrent
5027	Motor Failed: Servo Processor Alarm
5032	Motor Failed: Attempt of motion while a fault is active
5033	Motor Failed: Attempt of motion in disabled direction

Table 29 MERR Return Values (page 2 of 2)

Error Code	Description
5035	Motor Failed: Program Error
5036	Motor Failed: Memory Overuse
5037	Motor Failed: Time Overuse
5038	Motor Failed: Emergency Stop
5039	Motor Failed: Servo Interrupt
5060	(Integrated Control Models only) Drive alarm: No fault
5061	(Integrated models only) Drive alarm: Short Circuit
5062	(Integrated Control Models only) Drive alarm: External Protection Activated
5063	(Integrated Control Models only) Drive alarm: Power supply too low
5064	(Integrated Control Models only) Drive alarm: Power supply too high
5065	(Integrated Control Models only) Drive alarm: Temperature too high
5066	(Integrated Control Models only) Drive alarm: Power supply 24VF1
5067	(Integrated Control Models only) Drive alarm: Power supply 24VF2.
5068	(Integrated Control Models only) Drive alarm: Emergency Stop
5069	(Integrated models only) Drive alarm: Power down
5075	(Integrated Control Models only) Drive alarm: Digital drive interface not connected.

Tag

86

Comments

MERR is updated every time the axis motion is terminated. MERR stores the last termination code until either **FCLEAR** or **ENABLE/ENABLEALL** is executed.

Accessibility

Read-Only

Related ACSPL+ Commands

FCLEAR

Related ACSPL+ Variables

AERR, **PERR**

COM Library Methods

GetMotorError, FaultClear, FaultClearM

C Library Functions

acsc_GetMotorError, acsc_FaultClear, acsc_FaultClearM

3.8.6 SAFIN

Description

SAFIN is a 64 member (one for each axis) integer array with a set of bits that indicates the raw state, before processing, of the axis safety inputs.

The value of each element in the array ranges from -2147483648 to 2147483647, Default=0.

Tag

121

Comments

1. The **SAFIN** uses the same bit numbers as in **S_SAFIN** and as the corresponding faults in **FAULT** and **S_FAULT**.
2. **SAFIN** is normally read-only. However, when working with the Simulator, read/write is permitted to simulate safety inputs.
3. Only the **SAFIN** bits below are valid.

Bit Name	No.	Description
#RL	0	Right Limit
#LL	1	Left Limit
#HOT	4	Overheat
#DRIVE	9	Drive Alarm

Accessibility

Read-Only

Note	
	SAFIN *can be written to when working with the SPiiPlus Simulator.*

Related ACSPL+ Variables

FAULT, **S_FAULT**, **FDEF**, **S_FDEF**, **FMASK**, **S_FMASK**, **S_SAFIN**, **SAFINI**, **S_SAFINI**

COM Library Methods

SetSafetyInputPort

C Library Functions

acsc_SetSafetyInputPort

3.8.7 SAFINI

Description

SAFINI is a 64 member (one for each axis) integer array with a set of bits defining the active state of the **axis** safety input variable (**SAFIN**) specifying inversion of the signal input logic, if required.

Syntax

SAFINI(*axis_index)[.bit_designator] = value*

Arguments

axis_index	**axis_index** designates the specific axis: 0, 1, 2, .. 7.
bit_designator	The valid **SAFINI** bits are given in **Table 30**.
value	**value** ranges from -2147483648 to 2147483647, Default=0.

Table 30 SAFINI Valid Bits

Bit Name	No.	Description
#RL	0	Right Limit
#LL	1	Left Limit
#HOT	4	Overheat
#DRIVE	9	Drive Alarm

Tag

122

Comments

1. When a **SAFINI** bit=0, the corresponding signal is not inverted and the high voltage state is considered active.
2. When a **SAFINI** bit=1, the bit is inverted and the low voltage state is considered active.

Note	
	SAFINI *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

FAULT, **S_FAULT**, **FDEF**, **S_FDEF**, **FMASK**, **S_FMASK**, **SAFIN**, **S_SAFIN**, **S_SAFINI**.

COM Library Methods

SetSafetyInputPortInv, GetSafetyInputPortInv

C Library Functions

acsc_SetSafetyInputPortInv, acsc_GetSafetyInputPortInv

3.8.8 S_ERR

Description

S_ERR is a scalar integer that contains the code of the initialization error set during powerup.

Tag

113

Comments

Accessibility

Read-Only

Related ACSPL+ Variables

None

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.8.9 S_FAULT

Description

S_FAULT is a scalar integer variable consisting of a set of bits equating to the occurrence of faults. **S_FAULT** has two categories of bits, Axis Faults and System Faults (faults that are not related to any specific axis).

The **S_FAULT** bits are described in **Table 31.**

Table 31 S_FAULT Fault Bits (page 1 of 3)

Bit	Fault	Fault Description
Axis Faults		
0	#RL	RIGHT LIMIT. 1 = Right limit switch is activated.
1	#LL	LEFT LIMIT. 1 = Left limit switch is activated.
2	#RL2	RIGHT LIMIT 2. 1 = Preliminary right limit switch is activated.

Table 31 S_FAULT Fault Bits (page 2 of 3)

Bit	Fault	Fault Description
3	#LL2	LEFT LIMIT 2. 1 = Preliminary left limit switch is activated.
4	#HOT	MOTOR OVERHEAT. 1 = Motor's temperature sensor indicates overheat.
5	#SRL	SOFTWARE RIGHT LIMIT. 1 = Axis reference position (**RPOS**) is greater than the software right limit margin (SRLIMIT).
6	#SLL	SOFTWARE LEFT LIMIT. 1 = Axis reference position (**RPOS**) is less than the software left limit margin (SLLIMIT).
7	#ENCNC	ENCODER NOT CONNECTED. 1 = Primary encoder (for digital encoder type only) is not connected.
8	#ENC2NC	ENCODER 2 NOT CONNECTED. 1 = Secondary encoder (for digital encoder type only) is not connected.
9	#DRIVE	DRIVE ALARM. 1 = Signal from the drive reports a failure.
10	#ENC	ENCODER ERROR. 1 = Primary encoder miscounts.
11	#ENC2	ENCODER 2 ERROR. 1 = Secondary encoder miscounts.
12	#PE	POSITION ERROR. 1 = Position error (PE) has occurred. PE is defined by the following variables: ❑ **ERRI** - Maximum position error while the axis is idle ❑ **ERRV** - Maximum position error while the axis is moving with constant velocity ❑ **ERRA** - Maximum position error while the axis is accelerating or decelerating ❑ **DELI** - Delay on transition from **ERRA** to **ERRI** ❑ **DELV** - Delay on transition from **ERRA** to **ERRV**
13	#CPE	CRITICAL POSITION ERROR. 1 = Position error (#PE) exceeds the value of the critical limit. #CPE errors occur outside normal range of operation and #CPE > #PE. The critical limit depends on the axis state and is defined by the following variables: ❑ **CERRI** if the axis is idle (not moving) ❑ **CERRV** if the axis is moving with constant velocity ❑ **CERRA** if the axis is accelerating or decelerating ❑ **DELI** - Delay on transition from **ERRA** to **ERRI** ❑ **DELV** - Delay on transition from **ERRA** to **ERRV**
14	#VL	VELOCITY LIMIT. 1 = Absolute value of the reference velocity (**RVEL**) exceeds the limit defined by the **XVEL** parameter.

Table 31 S_FAULT Fault Bits (page 3 of 3)

Bit	Fault	Fault Description
15	#AL	ACCELERATION LIMIT. 1 = Absolute value of the reference acceleration (**RACC**) exceeds the limit defined by the **XACC** parameter.
16	#CL	CURRENT LIMIT. 1 = **RMS** current calculated in the Servo Processor exceeds the limit value defined by the **XRMS** parameter.
17	#SP	SERVO PROCESSOR ALARM. 1 = Axis Servo Processor loses its synchronization with the MPU. The fault indicates a fatal problem in the controller.
20	#HSSINC	HSSI NOT CONNECTED. 1 = HSSI module is not connected.
System Faults		
25	#PROG	PROGRAM FAULT 1 = Run time error occurs in one of the executing ACSPL+ programs.
26	#MEM	MEMORY FAULT 1 = User application requires too much memory.
27	#TIME	TIME OVERUSE 1 = User application consumes too much time in the controller cycle.
28	#ES	EMERGENCY STOP 1 = ES signal is activated.
29	#INT	SERVO INTERRUPT 1 = The servo interrupt that defines the controller cycle is not generated. The fault indicates a fatal controller problem.
30	#INTGR	INTEGRITY VIOLATION 1 = The integrity of the user application in controller RAM is checked by the controller at power-up and whenever an **#IR** Terminal command is issued.
31	#FAILURE	COMPONENT FAILURE 1 = An MC4U hardware component other than the drive, such as the Power Supply, I/O card, or encoder card, has failed.

Tag

114

Comments

An **S_FAULT** bit, such as Left Limit, will be = 1 whenever one or more Left Limit fault bits are = 1. In this manner, **S_FAULT** provides an indication of the aggregate state of each **FAULT** bit.

Accessibility

Read-Only

Related ACSPL+ Variables

FAULT, **FDEF**, **S_FDEF**, **FMASK**, **S_FMASK**, **SAFIN**, **S_SAFIN**, **SAFINI**, **S_SAFINI**

COM Library Methods

GetFault

C Library Functions

acsc_GetFault

3.8.10 S_FDEF

Description

S_FDEF is a scalar integer variable consisting of a set of bits for defining the default response for the system faults contained in **S_FAULT**. **S_FDEF** is connected to **S_FAULT** in the same way that **FDEF** is connected with **FAULT**.

Syntax

S_FDEF*[.bit_designator] = value*

Arguments

bit_designator	The **S_FDEF** bits and associated responses are given in **Table 32.**
value	**value** ranges from -2147483648 to 2147483647, Default = 1.

Table 32 S_FDEF Bit Description (page 1 of 2)

Bit	Fault	Fault Description	Default Response (S_FDEF)
25	#PROG	PROGRAM FAULT 1 = Run time error occurs in one of the executing ACSPL+ programs.	The controller kills all axes.
26	#MEM	MEMORY FAULT 1 = User application requires too much memory.	The controller kills all axes.
27	#TIME	TIME OVERUSE 1 = User application consumes too much time in the controller cycle.	No default response.
28	#ES	EMERGENCY STOP 1 = ES signal is activated.	The controller disables all axes, halts all buffer programs, and sets OFFSET of each axis to 0.
29	#INT	SERVO INTERRUPT 1 = The servo interrupt that defines the controller cycle is not generated. The fault indicates a fatal controller problem.	The controller disables all axes.

Table 32 S_FDEF Bit Description (page 2 of 2)

Bit	Fault	Fault Description	Default Response (S_FDEF)
30	#INTGR	INTEGRITY VIOLATION 1 = The integrity of the user application in controller RAM is checked by the controller at power-up and whenever an **#IR** Terminal command is issued.	No default response
31	#FAILURE	COMPONENT FAILURE 1 = An MC4U hardware component other than the drive, such as the Power Supply, I/O card, or encoder card, has failed.	No default response The user has to supply a user-defined fault response.

Tag

115

Comments

The default value for all **S_FDEF** bits is 1, which enables the default response. If an **S_FDEF** bit = 0, the default response is disabled.

Note	
	S_FDEF *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

FAULT, **S_FAULT**, **FDEF**, **FMASK**, **S_FMASK**, **SAFIN**, **S_SAFIN**, **SAFINI**, **S_SAFINI**

COM Library Methods

GetResponseMask, SetResponseMask, GetFaultMask, SetFaultMask

C Library Functions

acsc_GetResponseMask, acsc_SetResponseMask, acsc_GetFaultMask, acsc_SetFaultMask

3.8.11 S_FMASK

Description

S_FMASK is scalar integer variable consisting of a set of bits for enabling or disabling the system faults contained in **S_FAULT**. **S_FMASK** is connected to **S_FAULT** in the same way that **FMASK** is connected with **FAULT**.

Syntax

S_FMASK*[.bit_designator] = value*

Arguments

bit_designator	The **S_FMASK** bits and associated responses are given in **Table 33**.
value	**value** ranges from -2147483648 to 2147483647, Default=0.

Table 33 S_FMASK Bit Description

Bit	Fault	Fault Description
25	#PROG	PROGRAM FAULT 1 = Run time error occurs in one of the executing ACSPL+ programs.
26	#MEM	MEMORY FAULT 1 = User application requires too much memory.
27	#TIME	TIME OVERUSE 1 = User application consumes too much time in the controller cycle.
28	#ES	EMERGENCY STOP 1 = ES signal is activated.
29	#INT	SERVO INTERRUPT 1 = The servo interrupt that defines the controller cycle is not generated. The fault indicates a fatal controller problem.
30	#INTGR	INTEGRITY VIOLATION 1 = The integrity of the user application in controller RAM is checked by the controller at power-up and whenever a **#IR** immediate command is issued.
31	#FAILURE	COMPONENT FAILURE 1 = An MC4U hardware component other than the drive, such as the Power Supply, I/O card, or encoder card, has failed.

Tag

117

Comments

The S_FMASK default value = 1 and causes the controller to check for the fault associated with that bit, as follows:

0: The corresponding **FAULT** bit is disabled

1: The corresponding **FAULT** is enabled and examined each MPU cycle.

Note	
	S_FMASK *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

FAULT, **S_FAULT**, **FDEF**, **S_FDEF**, **FMASK**, **SAFIN**, **S_SAFIN**, **SAFINI**, **S_SAFINI**

COM Library Methods

GetFaultMask, SetFaultMask

C Library Functions

acsc_GetFaultMask, acsc_SetFaultMask

3.8.12 S_SAFIN

Description

S_SAFIN is a scalar integer variable that indicates the raw state of the **#ES** bit (Emergency Stop) input stored in the **SAFIN** variable.

The value ranges from -2147483648 to 2147483647, Default=0.

Tag

118

Comments

S_SAFIN uses the same bit numbers as in **SAFIN** and as the corresponding faults in **FAULT** and **S_FAULT**, but only the **#ES** bit is meaningful.

Accessibility

Read-Only

Note	
	S_SAFIN *can be written to when working with the SPiiPlus Simulator.*

Related ACSPL+ Variables

FAULT, **S_FAULT**, **FDEF**, **S_FDEF**, **FMASK**, **S_FMASK**, **SAFIN**, **SAFINI**, **S_SAFINI**

COM Library Methods

SetSafetyInputPort, GetSafetyInputPort

C Library Functions

acsc_SetSafetyInputPort, acsc_GetSafetyInputPort

3.8.13 S_SAFINI

Description

S_SAFINI is a scalar integer variable used for defining the active state of the **system** safety input variable (**S_SAFIN**) specifying inversion of the signal input logic, if required.

Tag

119

Comments

1. When a **S_SAFINI** bit=0, the corresponding signal is not inverted and the high voltage state is considered active.
2. When a **S_SAFINI** bit=1, the bit is inverted and the low voltage state is considered active.

Note	
	S_SAFINI *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Variables

FAULT, **S_FAULT**, **FDEF**, **S_FDEF**, **FMASK**, **S_FMASK**, **SAFIN**, **S_SAFIN**, **SAFINI**

COM Library Methods

SetSafetyInputPortInv, GetSafetyInputPortInv

C Library Functions

acsc_SetSafetyInputPortInv, acsc_GetSafetyInputPortInv

3.9 Induction Motor Variables

The Induction Motor variables are:

Name	Description
SLCFIELD	Induction Motor Excitation
SLCSLIP	Induction Motor Slip Factor

3.9.1 SLCFIELD

Description

SLCFIELD is a 64 member (one for each axis) real array. It is used along with **SLCSLIP**for controlling permanent magnet (PM) synchronous motors (so called “DC Brushless motors”). **SLCFIELD** defines the magnetic field component.

Syntax

SLCFIELD*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** provides the percentage of the peak current of the amplifier ranging from 0 to 25.

Tag

217

Comments

Vector control, also known as Field Oriented Control, is a control technique that imitates the DC motor operation and applies it to AC motors: PM synchronous motors (DC brushless motors) and induction motors. This technique decomposes the motor current into two independent components:

- The magnetizing (direct) component that influences the total magnetic flux.
- The torque-producing (quadrature) component that influences the generated torque. This component is perpendicular to the magnetizing component.

In PM synchronous motors, the magnetizing component is in phase with the permanent magnet field. The goal is usually to keep this component zero, so all the current is dedicated to torque production. This maximizes the torque/current ratio and improves the efficiency and dynamic performance.

The magnetic field variable, **SLCFIELD**, is usually set equal to the nominal magnetizing current of the motor. It should be around the “knee” of the magnetizing curve of the motor - it should be high enough to maximize the torque constant of the motor, but must not be too high to prevent magnetic saturation. The exact value is provided by the motor manufacturer, but it can also be estimated based on 10-40% of the nominal current.

The value of **SLCFIELD** is expressed as percentage of the peak current of the amplifier and is calculated according to the following formula:

$$Field = \frac{\sqrt{2} I_{mag}}{I_{peak}} 100$$

where:

I_{mag}	The nominal excitation current of the motor (rms value).	Provided by the manufacturer, of 10-40% of the nominal current
I_{peak}	The amplitude of the maximum current of the amplifier.	

Note	
	SLCFIELD = *0 identifies an induction motor.*

Accessibility

Read-Write

Related ACSPL+ Variables

SLCSLIP

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadInteger, acsc_WriteInteger

3.9.2 SLCSLIP

Description

SLCSLIP is a 64 member (one for each axis) real array. It is used along with **SLCFIELD**for controlling permanent magnet (PM) synchronous motors (so called “DC Brushless motors”). **SLCSLIP** defines the slip constant.

Syntax

SLCSLIP*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** designates the slip frequency ranging from 0 to 5000.

Tag

218

Comments

The range of **SLCSLIP** is typically 200-1000. A reference value is calculated as follows:

$$SLCSLIP = 131 \times f_n[Hz] \times s_n[\%] \times \frac{I_{peak}}{\sqrt{2} I_n}$$

where:

f_n - The nominal supply frequency

s_n - The nominal slip, given by:

$$s_n = \frac{n_s - n_n}{n_s}$$

where:

n_s - The synchronous velocity

n_n - The nominal velocity

I_{peak} - The peak current of the amplifier

I_n - The nominal motor current (rms)

For example, the nominal data of a 1.05kW motor is:

n_n = 2870rpm, n_s = 3000rpm, I_n = 8.1A, and *fn* = 50Hz

So the nominal slip of the motor is:

$$s = \frac{3000 - 2870}{3000} = 0.0433$$

For I_{peak} = 10A, the value of **SLCSLIP** will be:

$$SLCSLIP = 131 \times 50 \times 0.0433 \frac{10}{\sqrt{2} \times 8.1} = 248$$

Accessibility

Read-Write

Related ACSPL+ Variables

SLCFIELD

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadInteger, acsc_WriteInteger

3.10 Nanomotion Variables

The Nanomotion variables are used to set various Dead Zone parameters in support of Nanomotion piezo ceramic motor motion.

The Nanomotion variables are:

Name	Description
SLDZMIN	Defines the Dead Zone minimum position.
SLDZMAX	Defines the Dead Zone maximum position.
SLZFF	Defines the zero velocity feed forward position.

3.10.0.1 SLDZMIN

Description

SLDZMIN is a 64 member (one for each axis) real array used for defining the minimum position of the Dead Zone.

Syntax

SLDZMIN(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between -1.79769e+308 to 1.79769e+308, Default = 1.

Tag

162

Comments

The Dead Zone mechanism stops the motor when the position approaches the target within the value of **SLDZMIN.** The value depends on the system specifications; usually **SLDZMIN** is between 1.0 to 2.0 counts.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.10.0.2 SLDZMAX

Description

SLDZMAX is a 64 member (one for each axis) real array used for defining the maximum position of the Dead Zone.

Syntax

SLDZMIN(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between -1.79769e+308 to 1.79769e+308, Default = 1.

Tag

163

Comments

The Dead Zone mechanism starts the motor again when the error radius increases above the value **SLDZMAX**. The value depends on the system specifications; usually **SLDZMAX** is between 4.0 to 10.0 counts.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.10.0.3 SLZFF

Description

SLZFF is a 64 member (one for each axis) real array used for defining zero velocity feed forward position.

Syntax

SLZFF(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0 to 1.79769e+308, Default = 300.

Tag

189

Comments

Using **SLZFF** improves settling time by stopping the feed forward velocity when the axis is getting close to the target position. The distance from the target is defined by **SLZFF** (in user units). The proper value of **SLZFF** depends on the total moving mass and the resolution of the encoder. It increases with mass and encoder resolution. Usually:

- For HR1 motor with encoder resolution of 0.1μM, set **SLZFF** to 100: 300 counts.
- For HR8 motor with encoder resolution of 0.1μM, set **SLZFF** to 300 - 400 counts.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11 Servo-Loop Variables

Advanced	*Servo-Loop variables are fully accessible at the ACSPL+ level. While ACSPL+ programs generally do not refer to servo-loop variables at run time, an advanced program could change a servo-loop variable on the fly to provide adaptive control.*

The servo-loop variables are used for configuration and adjustment. The **Adjuster** feature of the SPiiPlus MMI (**SPiiPlus MMI Application Studio → Setup → Adjuster**) is a powerful, easy-to-use interface to the servo control loops.

The Servo-Loop variable is:

Name	Description
DCOM	Drive Command

Additional servo-loop variables are grouped as follows:

- **Servo-Loop Current Variables**
- **Servo-Loop Velocity Variables**
- **Servo-Loop Position Variables**
- **Servo-Loop Commutation Variables**
- **Servo-Loop Velocity Notch Filter Variables**
- **Servo-Loop Compensations Variables**
- **Servo-Loop Miscellaneous Variables**

3.11.1 DCOM

Description

DCOM is a 64 member (one for each axis) integer array used for defining the maximum drive command.

Syntax

DCOM*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from -100 to 100, Default = 0.

Tag

20

Comments

When operating in the open loop mode (**MFLAGS.1=1**), **DCOM** defines a percentage of the maximum drive command, for example, the SPiiPlus PCI 4/8 provides differential drive output from -10 to +10V. Therefore, assigning 100 to **DCOM** provides +10V on the drive output, -100 corresponds to -10V and 0 to 0V.

When operating in the closed loop mode (**MFLAGS.1=0**), **DCOM** defines an offset.

Note	
	DCOM *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

Related ACSPL+ Commands

MFLAGS

COM Library Methods

ReadVariable, Write Variable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.2 Servo-Loop Current Variables

The Servo-Loop Current variables are:

Name	Description
SLBIASA	Current phase A bias
SLBIASB	Current phase Bbias
SLIKI	Integrator gain
SLIKP	Integrator proportional gain

3.11.2.1 SLBIASA

Description

SLBIASA is a 64 member (one for each axis) real array used for defining the maximum controller voltage output.

Syntax

SLBIASA(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between -10 to 10, Default = 0.

Tag

149

Comments

SLBIASA is expressed as a percentage of the maximum controller voltage output.

1. For **integrated models**: **SLBIASA** is read-only and displays the measured value of the current input bias.
2. For **non-integrated models**: **SLBIASA** is read-write and specifies the bias of the drive output. The controller uses the value only for brushless motors commutated by the controller.

Accessibility

Read-Only (integrated models)

Read-Write (nonintegrated models)

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.2.2 SLBIASB

Description

SLBIASB is a 64 member (one for each axis) real array used for defining the maximum controller current.

Syntax

SLBIASB(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between -10 to 10, Default = 0.

Tag

150

Comments

SLBIASB is expressed as a percentage.

1. For **integrated models**: **SLBIASB** is read-only and displays the measured value of the current input bias.
2. For **nonintegrated models**: **SLBIASB** is read-write and specifies the bias of the drive output. The controller uses the value only for brushless motors commutated by the controller.

Accessibility

Read-Only (integrated models)

Read-Write (nonintegrated models)

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.2.3 SLIKI

Description

SLIKI is a 64 member (one for each axis) real array used for defining integrator coefficient for the current.

Syntax

SLIKI(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0 - 65000.

Tag

170

Comments

SLIKI is active only in integrated models.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.2.4 SLIKP

Description

SLIKP is a 64 member (one for each axis) real array used for defining proportional coefficient for the current.

Syntax

SLIKP(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0 - 256000, Default = 1000.

Tag

171

Comments

SLIKP is active only in integrated models.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.3 Servo-Loop Velocity Variables

The Servo-Loop Velocity variables are:

Name	Description
SLVKI	Sets velocity integrator coefficient
SLVKP	Sets the proportional velocity gain.
SLVLI	Integrator velocity limit

3.11.3.1 SLVKI

Description

SLVKI is a 64 member (one for each axis) real array used for specifying the velocity loop integrator coefficient.

Syntax

SLVKI(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0 to 20000; Default = 200.

Tag

179

Comments

None.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.3.2 SLVKP

Description

SLVKP is a 64 member (one for each axis) real array used for providing a proportional coefficient that is applied to the velocity of the specified axis.

Syntax

SLVKP(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0 to 16777215, Default = 100.

Tag

180

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.3.3 SLVLI

Description

SLVLI is a 64 member (one for each axis) real array used for providing an integrator limit for the velocity of the specified axis.

Syntax

SLVLI(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0 to 100, Default = 50.

Tag

181

Comments

SLVLI is expressed as a percentage of the maximum value.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.4 Servo-Loop Velocity Notch Filter Variables

Notch Filter variables serve for notching the servo-loop velocity filtering.

The Servo-Loop Velocity Notch Filter variables are:

Name	Description
SLVNFRQ	Notch filter frequency.
SLVNWID	Notch filter width.
SLVNATT	Notch filter attenuation.

3.11.4.1 SLVNFRQ

Description

SLVNFRQ is a 64 member (one for each axis) real array used for providing a notch filter frequency for the velocity of the specified axis.

Syntax

SLVNFRQ(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0.1 to 4000, Default = 300.

Tag

182

Comments

SLVNFRQ is expressed in Hz.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.4.2 SLVNWID

Description

SLVNWID is a 64 member (one for each axis) real array used for providing a notch filter width for the velocity of the specified axis.

Syntax

SLVNWID(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0.1 to 4000, Default = 700.

Tag

183

Comments

SLVNWID is expressed in Hz.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.4.3 SLVNATT

Description

SLVNATT is a 64 member (one for each axis) real array used for providing the attenuation of the notch frequency of the specified axis.

Syntax

SLVNATT(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0.05 to 20; Default = 50.

Tag

184

Comments

None.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.5 Servo-Loop Velocity Low Pass Filter Variables

Low Pass Filter variables serve for setting the velocity low pass filtering parameters.

The Servo-Loop Velocity Low Pass Filter variables are:

Name	Description
SLVSOF	Sets filter bandwidth
SLVSOFD	Sets filter damping

3.11.5.1 SLVSOF

Description

SLVSOF is a 64 member (one for each axis) real array used for providing a second order filter bandwidth for the velocity of the specified axis.

Syntax

SLVSOF(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0.1 to 4000, Default = 700.

Tag

185

Comments

SLVSOF is expressed in Hz.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.5.2 SLVSOFD

Description

SLVSOFD is a 64 member (one for each axis) real array used for providing a second order filter damping factor for the velocity of the specified axis.

Syntax

SLVSOFD(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0.3 to 1, Default = 0.707.

Tag

186

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.6 Servo-Loop Velocity Bi-Quad Filter Variables

Bi-Quad Filter variables serve for setting the velocity bi-quad filtering parameters.

The Servo-Loop Velocity Bi-Quad Filter variables are:

Name	Description
SLVB0DD	Sets the damping ratio denominator for a Bi-Quad filter.
SLVB0DF	Sets the denominator value of the Bi-Quad filter.
SLVB0ND	Sets the damping ratio numerator for a Bi-Quad filter.
SLVB0NF	Sets the numerator value of the Bi-Quad filter.

3.11.6.1 SLVB0DD

Description

SLVB0NF is a 64 member (one for each axis) real array used for setting the damping ratio denominator for a Bi-Quad filter to the velocity loop control in addition to the existing 2nd order Low-pass and Notch filters for the given axis.

Syntax

SLVB0DD*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** designates denominator value of the Bi-Quad damping ratio ranging from 0.1 to 1. See *SPiiPlus ACSPL+ Programmer Guide*.

Tag

208

Comments

The Bi-Quad filter is the most general 2nd order filter. It has two poles and two zeros. It can be thought of as a high-pass filter in series with a low-pass filter.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadInteger, acsc_WriteInteger

3.11.6.2 SLVB0DF

Description

SLVB0DF is a 64 member (one for each axis) real array used for setting the denominator value of the Bi-Quad filter algorithm applied to the velocity loop control in addition to the existing 2nd order Low-pass and Notch filters for the given axis.

Syntax

SLVB0DF*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** designates denominator value of the Bi-Quad filter algorithm ranging from 0.1 to 4000 [Hz]. See *SPiiPlus ACSPL+ Programmer Guide*.

Tag

209

Comments

The Bi-Quad filter is the most general 2nd order filter. It has two poles and two zeros. It can be thought of as a high-pass filter in series with a low-pass filter.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadInteger, acsc_WriteInteger

3.11.6.3 SLVB0ND

Description

SLVB0NF is a 64 member (one for each axis) real array used for setting the damping ratio numerator for a Bi-Quad filter to the velocity loop control in addition to the existing 2nd order Low-pass and Notch filters for the given axis.

Syntax

SLVB0ND*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** designates numerator value of the Bi-Quad damping ratio ranging from 0.1 to 1. See *SPiiPlus ACSPL+ Programmer Guide*.

Tag

210

Comments

The Bi-Quad filter is the most general 2nd order filter. It has two poles and two zeros. It can be thought of as a high-pass filter in series with a low-pass filter.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadInteger, acsc_WriteInteger

3.11.6.4 SLVB0NF

Description

SLVB0NF is a 64 member (one for each axis) real array used for setting the numerator value of the Bi-Quad filter algorithm applied to the velocity loop control in addition to the existing 2nd order Low-pass and Notch filters for the given axis.

Syntax

SLVB0NF*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** designates numerator value of the Bi-Quad filter algorithm ranging from 0.1 to 4000 [Hz]. See *SPiiPlus ACSPL+ Programmer Guide*.

Tag

211

Comments

The Bi-Quad filter is the most general 2nd order filter. It has two poles and two zeros. It can be thought of as a high-pass filter in series with a low-pass filter.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadInteger, acsc_WriteInteger

3.11.7 Servo-Loop Position Variables

The Servor-Loop Position variables are:

Name	Description
SLDRA	Defines disturbance rejection.
SLDRX	Defines the maximum DRA correction for a given axis

3.11.7.1 SLDRA

Description

SLDRA is a 64 member (one for each axis) real array used for defining the DRA frequency for the given axis.

The ACS proprietary Disturbance Rejection Algorithm (DRA) is used to improve the disturbance rejection response of the servo, and helps to minimize the position error during the settling phase and shorten the settling time.

Syntax

SLDRA*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** designates the DRA frequency ranging from 0 to 1500 [Hz].

Tag

206

Comments

The most common use of DRA is to improve the settling of systems mounted on passive isolation platforms. Passive isolation is typically used to isolate systems from disturbances transmitted from the floor. They employ a seismic mass supported on a soft spring made of rubber, metal, or air. The spring's damping action absorbs vibrations above the spring's resonance. For this reason, passive isolation manufacturers usually try to lower spring resonant frequency to increase the effective isolation range. When a servo force is applied to generate motion, it also acts on the isolated stationary base, causing it to vibrate. Because the frequency

is low (usually below 1 Hz, to 10 Hz) and damping is very light, the isolation system continues vibrating long after the motion profile has ended. This vibration acts as disturbance to the servo system, introduces position error, and extends the settling time.

The DRA is used to minimize the latter effect and improve the position error during settling.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadInteger, acsc_WriteInteger

3.11.7.2 SLDRX

Description

SLDRX is a 64 member (one for each axis) real array used for defining the maximum DRA correction for the given axis.

The ACS proprietary Disturbance Rejection Algorithm (DRA) is used to improve the disturbance rejection response of the servo, and helps to minimize the position error during the settling phase and shorten the settling time.

Syntax

SLDRX*(axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** designates the DRA correction ranging from 0 to 2^{23}.

Tag

207

Comments

The most common use of DRA is to improve the settling of systems mounted on passive isolation platforms. Passive isolation is typically used to isolate systems from disturbances transmitted from the floor. They employ a seismic mass supported on a soft spring made of rubber, metal, or air. The spring's damping action absorbs vibrations above the spring's resonance. For this reason, passive isolation manufacturers usually try to lower spring resonant frequency to increase the effective isolation range. When a servo force is applied to generate motion, it also acts on the isolated stationary base, causing it to vibrate. Because the frequency is low (usually below 1 Hz, to 10 Hz) and damping is very light, the isolation system continues vibrating long after the motion profile has ended. This vibration acts as disturbance to the servo system, introduces position error, and extends the settling time.

The DRA is used to minimize the latter effect and improve the position error during settling.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadInteger, acsc_WriteInteger

3.11.7.3 SLPKP

Description

SLPKP is a 64 member (one for each axis) real array used for setting the proportional coefficient of the position for the specified axis.

Syntax

SLPKP(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0 to 16777215, Default = 0.

Tag

175

Comments

Motor movement during commutation largely depends on the servo-loop parameters.

COMMUT will not operate properly if **SLPKP** is set to zero, or the integrator is very low.

Accessibility

Read-Write

Related ACSPL+ Commands

COMMUT

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.8 Servo-Loop Commutation Variables

The Servo-Loop Communtation variables are:

Name	Description
SLCHALL	Hall Shift
SLCNP	Number of Poles
SLCPA	Phase Advance
SLCOFFS	Commutation Offset
SLCORG	Commutation Origin
SLCPRD	Commutation Period

Note	
	The low-level variables in this chapter are normally not used by the user. *Generally, these variables are defined during the axis adjustment using* **SPiiPlus MMI Application Studio → Toolbox → Setup → Adjuster** *or the* **COMMUT** *command.*

3.11.8.1 SLCHALL

Description

SLCHALL is a 64 member (one for each axis) integer array that stores the Hall shift.

Tag

192

Comments

The **Adjuster** commutation program calculates this parameter and saves it.

Caution	
	Do not change this parameter manually.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.8.2 SLCNP

Description

SLCNP is a 64 member (one for each axis) real array that defines the number of poles for a rotary motor.

Syntax

SLCNP(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 2 to 1000, Default = 4.

Tag

152

Comments

For linear motors, set **SLCNP**=2.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.8.3 SLCPA

Description

SLCPA is a 64 member (one for each axis) real array that defines the servo-loop commutation phase advance in electrical degrees at **XVEL** (maximum allowed velocity for the motor).

Syntax

SLCPA(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 0 to 90, Default=0.

Tag

157

Comments

The actual phase advance is calculated as:

SLCPA*VEL/XVEL

where **VEL** is the current reference velocity.

Accessibility

Read-Write

Related ACSPL+ Variables

XVEL

3.11.8.4 SLCOFFS

Description

SLCOFFS is a 64 member (one for each axis) real array that defines a commutation offset in electrical degrees to be added to the commutation phase.

Syntax

SLCOFFS(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from -60 to 60, Default=0.

Tag

153

Comments

SLCOFFS defines **SLCOFFS** is valid only if a brushless motor is specified (**MFLAGS**(axis_index)**.#BRUSHL** = 1).

Assignment to **SLCOFFS** immediately changes the commutation phase. Use **SLCOFFS** to introduce a small correction to the commutation phase.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.8.5 SLCORG

Description

SLCORG is a 64 member (one for each axis) real array that defines the commutation phase in electrical degrees at the point of origin which is usually at the index point.

Syntax

SLCORG(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from: ❑ -60 to 60, Default=0 ❑ 0 to 360, Default=0

Tag

156

Comments

SLCORG is valid only if a brushless motor has been specified, i.e., **MFLAGS**(axis_index)**.#BRUSHL** = 1.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.8.6 SLCPRD

Description

SLCPRD is a 64 member (one for each axis) real array that defines the servo-loopcommutation period.

Syntax

SLCPRD(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 256 to 16777215, Default=8000.

Tag

158

Comments

SLCPRD defines the feedback counts per revolution for rotary motors and the feedback counts per two magnetic pitches for linear motors.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.9 Servo-Loop Compensations Variables

Servo-Loop Compensations variables are used to set various parameters that provide compensation to overcome certain motion problems.

The Servo-Loop Compensation variables are:

Name	Description
SLAFF	Defines acceleration feed forward for a given axis
SLFRC	Compensation for static friction
SLFRCD	Compensation for dynamic friction

3.11.9.1 SLAFF

Description

SLAFF is a 64 member (one for each axis) real array used for specifying the acceleration feed forward of the specified axis.

Syntax

SLAFF(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges from 0 to 16777215; Default: 0.

Tag

148

Comments

None.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.9.2 SLFRC

Description

SLFRC is a 64 member (one for each axis) real array used for specifying the velocity friction.

Syntax

SLFRC(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0 to 50, Default = 0.

Tag

167

Comments

SLFRC is expressed as a percentage of the maximum output.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.9.3 SLFRCD

Description

SLFRCD is a 64 member (one for each axis) real array used for providing dynamic friction compensation at the maximum velocity.

Syntax

SLFRCD(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0% to 5% of maximum command.

Tag

168

Comments

SLFRCD provides dynamic compensation at the maximum velocity **XVEL**. For lower velocities, the compensation is reduced proportionally with the velocity. The value of **SLFRCD** is given as a percentage (range is 0 to 50%) of the maximum command.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.10 Servo-Loop Miscellaneous Variables

This section contains a collection of miscellaneous variables employed for specific servo-loop purposes.

The Servo-Loop Miscellaneous variables are:

Name	Description
SLABITS	Total number of absolute position bits for an absolute encoder
SLPBITS	Number of bits in position feedback
SLIOFFS	Offset to be added to the result of the current loop control
SLCROUT	Commutation feedback routing , see
SLPROUT	Position feedback routing
SLVROUT	Velocity feedback routing
SLVRAT	Velocity feed forward ratio

3.11.10.1 SLABITS

Description

SLABITS is a 64 member (one for each axis) integer array used for setting the total number of absolute position bits for an absolute encoder.

Syntax

SLABITS(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 16 to 50, Default = 49.

Tag

220

Comments

SLABITS is used for setting the total number of absolute position bits for an absolute encoder. This is the sum of the multi-turn resolution bits and the turn resolution bits. For example, for an encoder with 17 bits turn resolution and 16 bits multi-turn resolution, **SLABITS** should be set to 33.

Accessibility

Read-Write

Related ACSPL+ Variables

E_TYPE

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.10.2 SLPBITS

Description

SLPBITS is a 64 member (one for each axis) integer array used for providing the number of bits in the position feedback for the specified axis.

Syntax

SLPBITS(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 2 to 16, Default = 14.

Tag

173

Comments

None.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.10.3 SLIOFFS

Description

SLIOFFS is a 64 member (one for each axis) real array used for offset to be added to the result of the current loop control.

Syntax

SLIOFFS(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between -50 to 50, Default = 0.

Tag

172

Comments

The variable contains value in percents of maximal drive output.

The primary goal of the variable is to compensate for an active component of the motor load. For example, in a vertical axis the weight of the carriage can be compensated.

The variable is valid for DC brush and brushless motors.

Normally, the variable is changed in the process of adjustment (use **SPiiPlus MMI Appication Studio → Toolbox → Setup → Adjuster Wizard**).

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.10.4 Feedback Routing

There are three variables that can be used for routing feedback:

Name	Description
SLCROUT	Commutation feedback routing
SLPROUT	Position feedback routing
SLVROUT	Velocity feedback routing

3.11.10.4.1 SLCROUT

Description

SLCROUT is a 64 member (one for each axis) real array used for setting the feedback routing of the velocity commutation for the specified axis.

Syntax

SLVROUT(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0 to 23, Default = 0. The values and the feedback sources associated with them are given in**Table 34**.

Table 34 SLCROUT Values

SLCROUT	FACC(dsp only)
0	According to E_TYPE velocity
001	From channel 0 quadrature velocity
002	From channel 0 SINCOS velocity
003	From channel 0 HSSI velocity
101	From channel 1 quadrature velocity
102	From channel 1 SINCOS velocity
103	From channel 1 HSSI velocity
201	From channel 2 quadrature velocity
202	From channel 2 SINCOS velocity
203	From channel 2 HSSI velocity
301	From channel 3 quadrature velocity
302	From channel 3 SINCOS velocity
303	From channel 3 HSSI velocity
X04	Rout from analog input X (where X can be 01 or 02)

Tag

Comments

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.10.4.2 SLPROUT

Description

SLPROUT is a 64 member (one for each axis) real array used for setting the feedback routing of the position for the specified axis.

Syntax

SLPROUT(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0 to 23, Default = 0. The values and the feedback sources associated with them are given in**Table 35**.

Table 35 SLPROUT Values

SLPROUT	FPOS
0	According to E_TYPE position
001	From channel 0 quadrature position
002	From channel 0 SINCOS position
003	From channel 0 HSSI position
101	From channel 1 quadrature position
102	From channel 1 SINCOS position
103	From channel 1 HSSI position
201	From channel 2 quadrature position
202	From channel 2 SINCOS position
203	From channel 2 HSSI position
301	From channel 3 quadrature position
302	From channel 3 SINCOS position
303	From channel 3 HSSI position
X04	Rout from analog input X (where X can be 01 or 02)

Tag

177

Comments

The controller supports a standard control loop configuration where 0 feedback position (**FPOS**) is obtained from the 0 encoder, **FPOS(1)** from the 1 encoder, etc.

SLPROUT ≠ 0 indicates **FPOS** is from an alternative sensor, for example, if **SLPROUT(0)** is 0104, **FPOS** is obtained from an analog input 0 rather than from the encoder. In this case, the feedback source could be a potentiometer or any other device that produces analog voltage proportional to the motor position.

The meaning of the routing value depends on the axis and the controller model. For example, a value of 1 specified for the 0 or 2 axis selects the 0 encoder, the same value for the 1 or 2 axis selects the 1 encoder.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.10.4.3 SLVROUT

Description

SLVROUT is a 64 member (one for each axis) real array used for setting the feedback routing of the velocity for the specified axis.

Syntax

SLVROUT(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between 0 to 23, Default = 0. The values and the feedback sources associated with them are given in**Table 36**.

Table 36 SLVROUT Values

SLVROUT	FVEL (dsp only)
0	According to E_TYPE velocity
001	From channel 0 quadrature velocity
002	From channel 0 SINCOS velocity
003	From channel 0 HSSI velocity
101	From channel 1 quadrature velocity
102	From channel 1 SINCOS velocity
103	From channel 1 HSSI velocity
201	From channel 2 quadrature velocity
202	From channel 2 SINCOS velocity
203	From channel 2 HSSI velocity
301	From channel 3 quadrature velocity
302	From channel 3 SINCOS velocity
303	From channel 3 HSSI velocity
X04	Rout from analog input X (where X can be 01 or 02)

Tag

Comments

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.11.10.5 SLVRAT

Description

SLVAT is a 64 member (one for each axis) real array used for defining velocity feed forward ratio.

Syntax

SLVAT(*axis_index) = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
value	**value** ranges between -8.38861e+006 to 8.38861e+006, Default = 1.

Tag

187

Comments

Velocity feed forward compensates for the velocity feedback, achieving zero position error at constant velocity.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.12 System Configuration Variables

The System Configuration variables are:

Name	Description
CFG	Configuration Mode
CTIME	Controller Cycle Time
ECHO	I/O Echo Channel
IENA	Interrupt Enable/Disable
IMASK	Interrupt Mask
ISENA	Interrupt-Specific Enable/Disable
S_FLAGS	System Flags

3.12.1 CFG

Description

CFG is an integer variable that indicates the application protection configuration mode.

Syntax

CFG = *value*

Arguments

value	**value** can be one of the following: 0: Controller is in protected mode 1: Controller is in normal mode

Tag

14

Comments

An attempt to assign a value to a protected variable when **CFG** = 0 causes an error.

Accessibility

Read-Only

COM Library Methods

ReadVariable

C Library Commands

acsc_ReadVariable

3.12.2 CTIME

CTIME is a scalar variable that defines the controller cycle time.

Syntax

CTIME = *value*

Arguments

value	**value** is 1.0 (milliseconds).

Tag

17

Comments

Many operations in the controller are synchronized to the controller cycle. For example, profile generation is executed each controller cycle.

Currently only 1.0 milliseconds is supported; faster times are planned for future NT versions.

If **CTIME** is used, before running, the program has to be saved to the controller and controller restarted.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.12.3 IENA

Description

IENA is a 23-bit mask variable used for enabling or disabling software and hardware interrupts from a specific source.

Syntax

IENA.*bit_designator = 1|0*

Arguments

bit_designator	The meanings of **bit_designator** are given in **Table 37**.

Table 37 IENA Bit Description

Bit	Interrupt
7	Enable MARK1 0 interrupt
8	Enable MARK2 0 interrupt
9	Enable MARK1 2 interrupt
10	Enable M2ARK2 2 interrupt
11	Enable MARK1 4 interrupt
12	Enable M2ARK2 4 interrupt
13	Enable MARK1 5 interrupt
14	Enable M2ARK2 5 interrupt
15	Enable Emergency Stop interrupt
16	Enable Physical motion end interrupt
17	Enable Logical motion end interrupt
18	Enable Motion failure (Motion interruption due to a fault) interrupt
19	Enable Motor failure (Motor disable due to a fault) interrupt
20	Enable Program termination interrupt
21	Enable Dynamic buffer interrupt
22	Enable ACSPL+ interrupt by **INTERRUPT** command
23	Enable Digital input interrupt
24	Enable Motion start interrupt
25	Enable Motion phase interrupt
26	Enable ACSPL+ interrupt by **TRIGGER** command

Tag

69

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.12.4 IMASK

Description

IMASK is a 64 member (one for each axis) integer array containing a set of bits that define which motor index and mark signals are processed.

Syntax

IMASK(*axis_index).bit_designator = value*

Arguments

axis_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
bit_designator	**IMASK** has four bit designators: ❑ **#IND** (bit 0) - Primary encoder index ❑ **#IND2** (bit 1) - Secondary encoder index ❑ **#MARK** (bit 2) - Mark1 ❑ **#MARK2** (bit 3) - Mark2
value	**value** ranges from -2147483648 to 2147483647, Default = 13

Tag

70

Comments

If a bit is zero, the controller neither analyzes or latches the corresponding **INDEX** or **MARK** signal.

Every axis does not provide all **INDEX** and **MARK** signals. The secondary encoder index is available only if a secondary encoder is used. **MARK** signals are only available for axes 0, 1, 4, and 5.

Note	
	IMASK *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

COM Library Methods

GetIndexState, ResetIndexState

C Library Functions

acsc_GetIndexState, acsc_ResetIndexState

3.12.5 ISENA

Description

ISENA is a 64 member (one for each axis) integer array containing a set of 8 bits used for enabling or disabling software interrupts within a specific interrupt status bit for a specific axis or buffer. Each element corresponds to one software interrupt status bit and specifies which axis, buffers or inputs are enabled to cause interrupt.

Syntax

ISENA*array_index.bit_designator = 1|0*

Arguments

array_index	Designates the specific axis, valid numbers are: 0, 1, 2, ... up to the number of axes in the system minus 1.
bit_designator	The meanings of **bit_designator** are given in **Table 38**.

Table 38 ISENA Bit Description

Bit	Interrupt
0	Controls Physical Motion End interrupt.
1	Controls Logical Motion End interrupt.
2	Controls Motion Failure interrupt.
3	Controls Motor Failure interrupt.
4	Controls Program Termination interrupt.
5	Controls Command Execution interrupt (dynamic buffer only).
6	Controls ACSPL+ interrupt (by **INTERRUPT** command).
7	Controls Digital Input interrupt.

Note	
	If the Physical Motion is PEG-related, then bit 0 is not supported in NT 1.0.

Tag

78

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.12.6 S_FLAGS

Description

S_FLAGS is an integer variable containing a set of bits that define different settings for the controller.

Syntax

S_FLAGS.*bit_designator = 1|0*

Arguments

bit_designator	The meanings of **bit_designator** are given in **Table 39**.

Table 39 S_FLAGS Bit Description (page 1 of 2)

Bit Name	No.	Description	Remarks
#S_FLAGS	1	**S_FLAGS.1** controls whether the controller allots a controller cycle to processing non-executable lines such as comments, empty lines and labels. 0: Comments, empty lines, labels are skipped during execution and not allotted a controller cycle. (Default) 1: These lines are each allotted a controller cycle.	Changes to **S_FLAGS.1** take effect only after a program is recompiled.
#FCLEAR	2	0: Sets the controller to regular mode. (Default) 1: Sets the controller to strict mode	In the regular mode the next motion command simply clears the reason for the previous **KILL**. In the strict mode, the next motion command cannot activate the motion and fails. Motion cannot be activated as long as the reason for the previous **KILL** is non-zero for any involved motor. Motion can continue only after clearing the **MERR** variable with **ENABLE/ENABLEALL** or **FCLEAR**.

Table 39 S_FLAGS Bit Description (page 2 of 2)

Bit Name	No.	Description	Remarks
#COPYFROM	5	0: Each controller cycle, the controller copies from the dual-port RAM to standard variables as defined by the last **DISP** command. (Default) 1: Copying is disabled	The **COPYFROMDPR** parameter definitions are retained and copying takes place whenever **S_FLAGS.#COPYFROM** = 0.
#COPYOIR	6	Determines the mode of **COPYFROMDPR** operation 0: Copy-each-cycle 1: Copy only if requested.	

Tag

116

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.13 Communication Variables

Communication variables are used for establishing various communication parameters.

The Communication variables are:

Name	Description
BAUD	Serial Communication Baud Rate
COMMCH	Communication Channel
COMMFL	Communication Flags
CONID	Controller Identification
DISPCH	Default Communication Channel
ECHO	Echo Communication Channel
ECST	Contains Ethernet status
ECERR	Contains Ethernet error code
TCPIP	IP Address for 1st Ethernet
TCPIP2	IP Address for 2nd Ethernet
TCPPORT	TCP port identifier
UDPPORT	UDP port identifier

3.13.1 BAUD

Description

BAUD is an integer variable that defines the serial communication rate, given in bits per second.

Syntax

BAUD = *value*

Arguments

value	**value** can be one of the following: ❑ 300 ❑ 1200 ❑ 4800 ❑ 9600 ❑ 19200 ❑ 57600 ❑ 115200

Tag

8

Comments

Changes to **BAUD** take effect only after controller restart.

Note	
	BAUD *values cannot be modified if protection is applied to this variable through* **SPiiPlus MMI Application Studio → Toolbox → Application Development → Protection**

Accessibility

Read-Write

COM Library Methods

OpenComSerial

C Library Commands

acsc_OpenComSerial

3.13.2 COMMCH

Description

COMMCH is an integer that stores a number representing the last activated communication channel. **Table 40**.

Table 40 COMMCH Values

Value	Description
1	Serial port 1
2	Serial port 2
6	Ethernet network (TCP)
7	Ethernet network (TCP)
8	Ethernet network (TCP)
9	Ethernet network (TCP)
10	Ethernet Point-to-Point (UDP)
12	PCI bus
16	MODBUS Slave
36	Ethernet network (TCP)
37	Ethernet network (TCP)
38	Ethernet network (TCP)
39	Ethernet network (TCP)

Tag

15

Comments

When queried through a communication channel, **COMMCH** reads the number of the current communication channel.

COMMCH can be used in **SEND**, or assigned to **DISPCH**.

Accessibility

Read-Only.

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.13.3 COMMFL

Caution	
	This variable is for advanced users. Changing the default values of these bits is not recommended!

Description

COMMFL is a scalar variable containing a set of bits that affect controller communication.

Syntax

COMMFL.*bit_designator = 1|0*

Arguments

bit_designator	The **COMMFL** bits and the meanings of their values are given in **Table 41**.

Table 41 COMMFL Bit Descriptions

Bit Name	No.	Description
#VERBOSE	0	Controls error message form. 0 = The controller provides the error number only to the C Library function or COM method, when an error occurs. 1 = The controller provides an extended message
	1	1 = Enable motor messages
	2	1 = Enable Axis messages
	3	1 = Enable Program messages
#SAFEMSG	4	1 = Controller sends unsolicited messages in Safe communication format
#CSUMMSG	6	1 = A checksum is included in unsolicited messages. Normally the user does not need to change this bit.
#NOCOMM	7	Controls the communication in protected mode. 1 = The controller ignores any command received via communication channels except the queries that start from '?' character. The bit is not effective if the controller is in configuration mode. The default value is 0.
#NOQUERY	8	Controls the communication in protected mode. 1 = The controller ignores any query received via communication channels. The bit is not effective if the controller is in configuration mode. The default value is 0.

Tag

16

Comments

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Commands

acsc_ReadVariable, acsc_WriteVariable

3.13.4 CONID

Description

CONID is an integer variable that contains the controller identification.

Syntax

CONID = *value*

Arguments

value	An integer ranging between 0 and 65536.

Tag

193

Comments

The controller identification can be used for many different purposes like MODBUS Slave ID, CAN Slave ID or user-defined unique ID within the user network.

Note	
	According to the MODBUS specification, the controller must have an individual address from 1 to 247. *In order to specify the MODBUS Slave address,* **CONID** *should be initialized to the MODBUS Slave address value.*

By default, the variable has zero value.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Commands

acsc_ReadVariable, acsc_WriteVariable

3.13.5 ECHO

Description

ECHO is a 10-member mask integer variable that defines an echo communication channel.

Syntax

ECHO = *channel_number*

Arguments

channel_number	The values of **channel_number** and their meanings are given in **Table 42**.

Table 42 ECHO Channel Numbers

Channel Number	Channel Name
-1	Echo NOT active
-2	All channels
1	Serial port 1
2	Serial port 2
6	Ethernet network (TCP)
7	Ethernet network (TCP)
8	Ethernet network (TCP)
9	Ethernet network (TCP)
10	Ethernet Point-to-Point (UDP)
12	PCI bus
16	MODBUS Slave
36	Ethernet network (TCP)
37	Ethernet network (TCP)
38	Ethernet network (TCP)
39	Ethernet network (TCP)

Tag

35

Comments

If **ECHO** specifies a valid communication channel, the controller sends an echo of each command received from any communication channel to the specified channel. Address each channel as follows:

The default value for **ECHO** is -1, in order to select an echo channel, the user needs to select a channel number.

ECHO cannot be saved to flash. After power-up the value is set to -1.

Use **DISPCH** to configure the communication channels related to the controller. Use **COMMCH** to retrieve the current controller channel's physical connection (only the channel that is connected to the terminal on which the query is sent).

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.13.6 ECST

Description

ECST is a scalar variable affecting the EtherCAT state. The EtherCAT state is reflected in the first six bits as given in **Table 43**.

Table 43 ECST Bits

Bit	Designator	Description
0	#SCAN	The scan process was performed successfully.
1	#CONFIG	There is no deviation between XML and actual setup.
2	#INITOK	All bus devices are successfully set to INIT state.
3	#CONNECTED	Indicates valid Ethernet cable connection to the master.
4	#INSYNC	If DCM is used, indicates synchronization between master and the bus.
5	#OP	The EtherCAT bus is operational.
6	#DCSYNC	Distributed clocks are synchronized.

Syntax

ECST.*bit_designator = 1|0*

Arguments

None

Comments

All bits (except #INSYNC in some cases) should be true for proper bus functioning.

For monitoring the bus state, checking bit #OP is sufficient. Any bus error will reset the #OP bit.

COM Library Methods
ReadVariable, WriteVariable

C Library Functions
acsc_ReadInteger, acsc_WriteInteger

3.13.7 ECERR

Description
ECERR is a scalar variable containing an EtherCAT error code. The EtherCAT error codes are given in **Table 44**.

Table 44 EtherCAT Error Codes

Error Code	Description
6000	General EtherCAT error.
6001	EtherCAT cable not connected.
6002	EtherCAT master is in incorrect state.
6003	Not all EtherCAT frames can be processed.
6004	EtherCAT Slave error.
6005	EtherCAT initialization failure.
6006	EtherCAT cannot complete the operation.
6007	EtherCAT work count error.
6008	Not all EtherCAT slaves are operational.
6009	EtherCAT protocol timeout.
6010	Slave initialization failed.
6011	Bus configuration mismatch

Syntax
ECERR

Arguments
None

Comments
Any EtherCAT error sets **ECST.#OP** to false and the error code is latched in **ECERR**.

COM Library Methods
ReadVariable

C Library Functions
acsc_ReadInteger

3.13.8 DISPCH

Description

DISPCH is a scalar integer variable that defines a communication channel between the controller and a host application, MMI or any device connected to the controller's communication ports.

Syntax

DISPCH = *channel_number*

Arguments

channel_number	The values of **channel_number** and their meanings are given in **Table 45**.

Table 45 DISPCH Channel Numbers

Channel Number	Description
-2	All channels
-1	No default channel is specified, the command uses the last channel activated by the host.
1	Serial port 1
2	Serial port 2
6	Ethernet network (TCP)
7	Ethernet network (TCP)
8	Ethernet network (TCP)
9	Ethernet network (TCP)
10	Ethernet Point-to-Point (UDP)
12	PCI bus
16	MODBUS Slave
36	Ethernet network (TCP)
37	Ethernet network (TCP)
38	Ethernet network (TCP)
39	Ethernet network (TCP)

Tag

25

Comments

DISPCH is relevant only to messages sent with **DISP** and **SEND** (described also as "Unsolicited Messages").

In order to view unsolicited messages in the **Terminal** window, select the check box in the lower right corner of the Terminal window to enable **Show Unsolicited Messages**.

If **DISPCH** specifies a valid communication channel, all unsolicited messages (messages that are sent with **DISP** and **SEND** from the program buffers) are sent to this channel irrespective of the channel used for immediate commands.

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

3.13.9 TCPIP

Description

TCPIP is an integer variable used for setting the TCP/IP for the Ethernet port N1.

Syntax

TCPIP = *value*

Arguments

value	**value** an hexidecimal number - format: 0xAABBCCDD.

Tag

133

Comments

If **TCPIP** has a non-zero value, the controller uses the value as its TCP/IP address. In this case, other configuration parameters receive the following default values:

- Subnet mask - 255.255.255.0
- Gateway address - no gateway, i.e., no routing is supported

If **TCPIP** is zero, the controller uses the DHCP protocol to receive the network configuration from the DHCP server. The network configuration received from the DHCP server includes the following parameters:

- Controller's TCP/IP address
- Subnet mask
- Gateway address

The **TCPIP** variable value has to be in hex, for example:

```
TCPIP=0x6400000a
```

assigns a TCP/IP address of: 10.0.0.100. Note that the address is calculated starting from the least significant byte of the value.

To retrieve the assigned address in an ACSPL+ program, use the **GETCONF** function with key 310.

Accessibility

Read-Write

C Library Commands

acsc_GetEthernetCards (to find all SPiiPlus controllers in the network segment)

3.13.10 TCPIP2

Description

TCPIP2 is an integer variable used for setting the TCP/IP for a second Ethernet port N2.

Syntax

TCPIP2 = *value*

Arguments

value	**value** an hexidecimal number - format: 0xAABBCCDD.

Tag

198

Comments

If **TCPIP2** is zero, the address will be automatically obtained at the controller start-up through DHCP protocol. The default address for second Ethernet port is 192.168.0.100.

The **TCPIP2** variable value has to be in hex: 0xAABBCCDD, for example:

TCPIP=0x6400000a

assigns a TCP/IP address of: 10.0.0.100. Note that the address is calculated starting from the least significant byte of the value.

To retrieve the assigned address in an ACSPL+ program, use the **GETCONF** function with key 310.

Accessibility

Read-Write

C Library Commands

acsc_GetEthernetCards (to find all SPiiPlus controllers in the network segment)

3.13.11 TCPPORT

Description

TCPPORT is a scalar integer that stores a number representing a TCP port.

Syntax

TCPPORT = *Port_number*

Arguments

Port_number	An integer ranging between 0 and 65536.

Tag

200

Accessibility

Read-Write

Comments

TCPPORT defines Ethernet ports in the controller for TCP. By default, this variable is set to 701. In order to establish communication with the controller through a port different from default port numbers, the following should be done:

1. Set **TCPPORT** to a value other than 701.

Note	
	Some of the ports are used by the controller firmware and cannot be used. It is recommended to use ports starting from 1024.

2. Save system parameters to the flash.
3. Restart the controller.
4. Try to establish communication using new ports by providing them in client user application. If communication isn't established, try to set other values.

Note	
	This new port value is used by the client user application only. The SPiiPlus Tools and SPiiPlus C/COM Library continue to use the default ports.

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadInteger, acsc_WriteInteger

3.13.12 UDPPORT

Description

UDPPORT is a scalar integer that stores a number representing a UDP port.

Syntax

UDPPORT = Port_number

Arguments

Port_number	An integer ranging between 0 and 65536.

Tag

201

Accessibility

Read-Write

Comments

UDPPORT defines Ethernet ports in the controller for UDP. By default, it is set to 700. In order to establish communication with the controller through different from default port numbers, the following should be done:

1. Set **UDPPORT** to a value other than 700.

Note	
	Some of the ports are used by the controller firmware and cannot be used. It is recommended to use ports starting from 1024.

2. Save system parameters to the flash.
3. Restart the controller.
4. Try to establish communication using new ports by providing them in client user application. If communication isn't established, try to set other values.

Note	
	This new port value is used by the client user application only. The SPiiPlus Tools and SPiiPlus C/COM Library continue to use the default ports.

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadInteger, acsc_WriteInteger

3.14 Miscellaneous

The Miscellaneous variables are:

Name	Description
FK	Function Key
XARRSIZE	Maximum Array Size

3.14.1 FK

Description

FK is a scalar integer and is used by the controller to store function keys in an input string processed with the **INPUT** command.

Tag

50

Comments

If the controller encounters a zero character in the input string, it stores the value of the next character in **FK**.

This is the usual way for loading function key codes (F1 - F12).

Note	
	An autoroutine can be used to respond to changes in **FK**.

Accessibility

Read-Only

COM Library Methods

ReadVariable

C Library Functions

acsc_ReadVariable

3.14.2 XARRSIZE

Description

XARRSIZE is a scalar integer that stores maximum size of an array.

Syntax

XARRSIZE = *value*

Arguments

value	**value** ranges from 10,000 to 10,000,000 (elements); Default = 100,000

Tag

219

Comments

By default the maximum size for a user array is 100,000 elements; if, however, an application requires larger arrays, the user may change this value to accommodate a larger array size. However the following should be taken into consideration:

1. Defining large arrays may use too much memory and may cause an out of memory fault. To avoid this, the RAM size available for user data in the specific controller model should be checked. One element in an array requires 8 bytes of RAM, 131,072 elements require 1 MB.
2. The processing time required for operations on large arrays in ACSPL+ may cause an over usage fault. Therefore, arrays should be defined only with the size actually required for the application.

Note	
	It strongly recommended that users change the **XARRSIZE** *variable only if necessary, and that such changes be tested under safe conditions.*

Accessibility

Read-Write

COM Library Methods

ReadVariable, WriteVariable

C Library Functions

acsc_ReadVariable, acsc_WriteVariable

4 ACSPL+ Functions

ACSPL+ functions are divided into the following categories:

- **Arithmetical Functions**
- **Array Processing Functions**
- **Miscellaneous Functions**
- **CoE Functions**
- **Servo Processor Functions**
- **Signal Processing Functions**

This chapter covers the ACSPL+ functions.

The ACSPL+ Functions, in alphabetical order, are:

Function	Action
ABS	Calculates the absolute value.
ACOS	Calculates the arc cosine.
ASIN	Calculates the arc sine.
ATAN	Calculates the arc tangent.
ATAN2	Calculates the arc tangent of X/Y.
AVG	Finds the average of all values in an array.
BCOPY	Copies a given number of bytes from a source array into a target array.
CEIL	Calculates the ceiling of a value.
COPY	The function copies one array to another.
COEREAD	Read CoE slave Object Directory entry.
COS	Calculates the cosine.
COEWRITE	Write into CoE slave Object Directory.
DEADZONE	Implements dead-zone routine.
DSIGN	Implements a dynamic version of the standard **SIGN** function.
DSTR	Converts a string to an integer array.
ECIN	Copy EtherCAT offset data into ACSPL+ variable
ECOUT	Copy data of ACSPL+ variable into EtherCAT offset
ECUNMAP	Undoes any results from running **ECIN** and **ECOUT**
EDGE	Returns 1 on positive edge of x.
EXP	Calculates the exponent.
FLOOR	Calculates the floor of a value.
GETCONF	Reads hardware and firmware parameters.
GETSP	Reads a value from the specified SP address.
GETSPA	Retrieves address of the SP variable specified by name.
GETSPV	Reads a value from the specified SP variable name.

Function	Action
HYPOT	Calculates the hypotenuse.
INP	Reads data characters from the specified channel and stores them into integer array.
INTGR	Implements an integrator with **DEADZONE** and **SAT**.
LAG	Provides delayed switching on argument change (anti-bouncing effect).
LDEXP	Calculates a value of x*2exp.
LOG	Calculates the natural logarithm.
LOG10	Calculates the base-10 logarithm.
MAP	Implements a table-defined function with constant step.
MAPB	One-dimensional uniform b-spline.
MAPN	One-dimensional non-uniform linear interpolation (replaces obsolete **MAPBY1** and **MAPBY2**).
MAPNB	One-dimensional non-uniform B-spline.
MAPNS	One-dimensional non-uniform Catmull-Rom spline.
MAPS	One-dimensional uniform Catmull-Rom spline.
MAP2	Implements a table-defined function with two arguments and constant step along each argument.
MAP2B	Two-dimensional uniform B-spline.
MAP2N	Two-dimensional non-uniform linear interpolation (replaces obsolete **MAP2FREE**).
MAP2NB	Two-dimensional non-uniform B-spline.
MAP2NS	Two-dimensional non-uniform Catmull-Rom spline.
MAP2S	Two-dimensional uniform Catmull-Rom spline.
MATCH	Calculates axis position that matches current reference position of the same axis with zero offset.
MAX	Finds the maximum value in an array.
MAXI	Finds the element with maximum value in an array or in a section of an array and returns its index.
MIN	Finds the minimum value in an array.
MINI	Finds the element with minimum value in an array or in a section of an array and returns its index.
MONSP	Initiates monitoring of the specified SP address through analog output.
MONSPV	Initiates monitoring of the specified SP variable name through analog output.
NUMTOSTR	Converts a number to an ASCII string.
POW	Calculates x raised to the power of y.
RAND	Implements a random number generator.
ROLL	Calculates a result rolled-over to within one pitch.
SAT	Implements a saturation characteristic.
SETCONF	Writes hardware and firmware parameters.

Function	Action
SETSP	Sets a value for the specified SP address.
SETSPV	Sets a value for the specified SP variable name.
SETVAR	Writes a value to a system or user variable, scalar or array, that was declared as a Tag number.
SIGN	Returns -1, 0 or 1 depending on the sign of x.
SIN	Calculates the sine.
SQRT	Calculates the square root.
STR	Converts an integer array to a string.
STRTONUM	Converts an ASCII string representing a number to the number it represents.
SYSINFO	Returns certain system information based on the argument that is specified.
TAN	Calculates the tangent.

4.1 Arithmetical Functions

The Arithmetical functions are:

Function	Action
ABS	Calculates the absolute value
ACOS	Calculates the arc cosine
ASIN	Calculates the arc sine
ATAN	Calculates the arc tangent
ATAN2	Calculates the arc tangent of Y/X
CEIL	Calculates the ceiling of a value
COS	Calculates the cosine
EXP	Calculates the exponent
FLOOR	Calculates the floor of a value
HYPOT	Calculates the hypotenuse
LDEXP	Calculates a value of x*2exp
LOG	Calculates the natural logarithm
LOG10	Calculates the base 10 logarithm
POW	Calculates x raised to the power of y
SIGN	Returns -1, 0 or 1 depending on the sign of x
SIN	Calculates the sine
SQRT	Calculates the square root
TAN	Calculates the tangent

4.1.1 ABS

Description
ABS calculates the absolute value

Syntax
ABS*(input)*

Arguments

input	**input** can be a real number or an expression.

Return Value
Real number - returns the absolute value of the input.

Error Conditions
None

Example
```
XX = ABS(-3.14)
DISP XX !Output = 3.14
```

4.1.2 ACOS

Description
ACOS calculates arc cosine.

Syntax
ACOS(*input*)

Arguments

input	**input** can be a real number or an expression.

Return Value
Real number - returns the arc cosine of the argument in the range from 0 to π radians.

Error Conditions
The value of **input** must be between –1 to 1, otherwise the function returns Error 3045, Numerical Error in Standard Function.

Example
```
XX = ACOS(-1)
DISP XX !Output = 3.141592654
```

4.1.3 ASIN

Description
ASIN calculates the arc sine.

Syntax
ASIN(*input*)

Arguments

input	**input** can be a real number or an expression.

Return Value
Real number - returns the arc sine of XX in the range –π/2 to π/2.

Error Conditions
The value of **input** must be between –1 to 1, otherwise the function returns Error 3045, Numerical Error in Standard Function.

Example
```
XX = ASIN(-1)
DISP XX !Output = -1.570796327
```

4.1.4 ATAN

Description
ATAN calculates the arc tangent.

Syntax
ATAN(*input*)

Arguments

input	**input** can be a real number or an expression.

Return Value
Real number - returns the arc tangent of the input in the range –π/2 to π/2 radians.

Error Conditions
None

Example
```
XX = ATAN(-1)
DISP XX !Output = –0.7853981634
```

4.1.5 ATAN2

Description
ATAN2 calculates the arc tangent of **X/Y**.

Syntax
ATAN2(*X_input*,*Y_input*).

Arguments

X_input	**X_input** can be a real number or an expression.
Y_input	**Y_input** can be a real number or an expression.

Return Value
Real number - returns the arc tangent value of **X_input**, **Y_input. ATAN2** calculates a value in the range of $-\pi$ to π radians using the signs of both parameters to determine the quadrant of the return value. If both parameters are 0, the function returns 0. **ATAN2** is well defined even if Y equals 0.

Error Conditions
None

Example
```
X_input = -1; Y_input = 0
XX=ATAN2(X_input,Y_input)
DISP XX !Output = -1.5708
```

4.1.6 CEIL

Description
CEIL calculates the ceiling of a value.

Syntax
CEIL(*input*)

Arguments

input	**input** can be a real number or an expression.

Return Value
Integer number - returns a value that represents the smallest integer that is ≥ **input**.

Error Conditions
None

Example
```
XX=CEIL(3)
YY=CEIL(-3)
ZZ=CEIL(2.1)
TT=CEIL(–2.1)
DISP XX, YY, ZZ, TT !Output = 3 -3 3 -2
```

4.1.7 COS

Description
COS calculates the cosine.

Syntax
COS(*input*)

Arguments

input	**input** can be a real number or an expression (which is treated as radians by the function).

Return Value
Real number - returns the cosine of X in the range of -1 to 1.

Error Conditions
None

Example
```
XX = COS(-3.141592654)
DISP XX !Output = -1
```

4.1.8 EXP

Description
EXP calculates the e^ input

Syntax
EXP(*input*)

Arguments

input	**input** can be a real number or an expression.

Return Value
Real number - returns the exponential value of **input**. On overflow, the function returns the largest real number.

Error Conditions
None

Example
```
XX = EXP(1)
DISP XX !Output = 2.718281828
```

4.1.9 FLOOR

Description
FLOOR calculates the floor of a value.

Syntax
FLOOR(*input*)

Arguments

input	**input** can be a real number or an expression.

Return Value
Integer number - returns a value representing the largest integer that is $\leq$ to **input**.

Error Conditions
None

Example
```
XX=FLOOR(3)
YY=FLOOR(-3)
ZZ=FLOOR (2.1)
TT=FLOOR(-2.1)
DISP XX,YY,ZZ,TT !Output = 3 -3 2 -3
```

4.1.10 HYPOT

Description
HYPOT calculates the hypotenuse of a right triangle

Syntax
HYPOT(*X_input*, *Y_input*)

Arguments

X_input	**X_input** can be a real number or an expression.
Y_input	**Y_input** can be a real number or an expression.

Return Value
Real number - calculates the length of the hypotenuse of a right triangle, given the length of the two sides **X_input** and **Y_input**. **HYPOT** is equivalent to the square root of $X^2 + Y^2$.

Error Conditions
None

Example
```
XX=HPOT(3,4)
DISP XX !Output = 5
```

4.1.11 LDEXP

Description
LDEXP calculates the value of x*2^exp.

Syntax
LDEXP(*X_input*,*Y_input*)

Arguments

X_input	**X_input** can be a real number or an expression.
Y_input	**Y_input** can be a real number or an expression.

Return Value
Real number - returns the value of **X_input***$2^{\mathbf{Y_input}}$. On overflow **LDEXP** returns the largest real number with a sign, depending on the sign of **X_input**.

Error Conditions
None

Example
```
XX= LDEXP(1,2)
DISP XX !Output = 4
```

4.1.12 LOG

Description
LOG calculates the natural logarithm.

Syntax
LOG(*input*)

Arguments

input	**input** can be a real number or an expression.

Return Value
Real number - returns the natural logarithm of **input**.

Error Conditions
input must be > 0, otherwise the function returns Error 3045, Numerical Error in Standard Function.

Example
```
XX=LOG(2.718281829)
DISP XX !Output = 1
```

4.1.13 LOG10

Description
LOG10 calculates the base 10 logarithm.

Syntax
LOG10(*input*)

Arguments

input	**input** can be a real number or an expression.

Return Value
Real number - returns the base 10 logarithm of **input**.

Error Conditions
input must be ≤ 0, otherwise the function returns Error 3045, Numerical Error in Standard Function.

Example
```
XX=LOG10(10)
DISP XX !Output = 1
```

4.1.14 POW

Description
POW calculates X raised to the power of Y.

Syntax
POW(*X_input*,*Y_input*)

Arguments

X_input	**X_input** can be a real number or an expression.
Y_input	**Y_input** can be a real number or an expression.

Return Value
Real number - returns the value of $\mathbf{(X_input)^{Y_input}}$.

Error Conditions
None

Example
```
XX=POW(2,3)
DISP XX !Output = 8
```

4.1.15 SIGN

Description
SIGN returns –1, 0 or 1 depending if the input is negative, zero or positive.

Syntax
SIGN(*input*)

Arguments

input	**input** can be a real number or an expression.

Return Value
Real number - returns:

–1 if **input** <0;

0 if **input** = 0;

1 if ***input*** >0

Error Conditions
None

Example
```
XX=SIGN(-5), SIGN(0),SIGN(5)
DISP XX !Output = -1 0 1
```

4.1.16 SIN

Description
SIN calculates the sine.

Syntax
SIN(*input*)

Arguments

input	**input** can be a real number or an expression (which is treated as radians by the function).

Return Value
Real number - returns the sine value of **input** in the range of –1 to 1.

Error Conditions
None

Example
```
XX=SIN(1.570796327)
DISP XX !Output = 1
```

4.1.17 SQRT

Description
SQRT calculates the square root.

Syntax
SQRT(*input*)

Arguments

input	**input** can be a real number or an expression.

Return Value
Real number - returns the square root of **input**.

Error Conditions
input must be > 0, otherwise the function returns Error 3045, Numerical Error in Standard Function.

Example
```
XX=SQRT(4)
DISP XX !Output = 2
```

4.1.18 TAN

Description
TAN calculates the tangent.

Syntax
TAN(*input*)

Arguments

input	**input** can be a real number or an expression (which treated as radians by the function).

Return Value
Real number - returns the tangent value of **input**.

Error Conditions
None

Example
```
REAL PI
PI = 3.141592654
DISP TAN(PI/4) !Output = 1
```

4.2 Array Processing Functions

The Array Processing functions are:

Function	Action
AVG	Finds the average of all values in an array.
COPY	Copies data from one user array to another.
FILL	Fills an array or a section of array with the specified value.
MAX	Finds the maximal value in an array
MAXI	Finds the maximal value in an array or in a section of array and returns its index.
MIN	Finds the minimal value in an array.
MINI	Finds the element with minimal value in an array or in a section of an array and returns its index.

4.2.1 AVG

Description
AVG finds the average of all values in an array.

Syntax
AVG(*array_name*)

Arguments

array_name	The name of an array that has been declared in the program.

Return Value
Real number - returns the average of all elements in **array_name**.

Example
```
REAL Ar(3)
Ar(0) = 1; AR(1)=0.5; Ar(2)=3;
DISP AVG(Ar) !Output = 1.5
```

4.2.2 COPY

Description

COPY copies data from one user array to another.

Syntax
COPY(*source, destination, from_source_row, to_source_row, from_source_col, to_source_col, from_destination_row, to_destination_row ,from_destination_col, to_destination_col*)

Arguments

source	The name of an array that has been declared in the program from which the data is to be copied.
destination	The name of an array that has been declared in the program to which the data is to be copied.
from_source_row	The index of the first row of the source to begin copying.
to_source_row	The index of the last row of the source to end copying.
from_source_col	The index of the first column of the source to begin copying. Used only for matrix type arrays, otherwise it can be ommitted.
to_source_col	The index of the last column of the source to end copying. Used only for matrix type arrays, otherwise it can be ommitted.
from_destination_row	The index of the first row of the destination to begin copying into.
to_destination_row	The index of the last row of the destination to end copying into.
from_destination_col	The index of the first column of the destination to begin copying into. Used only for matrix type arrays, otherwise it can be ommitted.
to_destination_col	The index of the last column of the destination to begin copying into. Used only for matrix type arrays, otherwise it can be ommitted.

Comments

If the matrix indexes are omitted, the entire source matrix will be copied to the destination matrix.

If the destination matrix has different dimensions than the source matrix then the destination matrix will use the source matrix values to fill each row completely and move to the next row.

Return Value

None

Error Conditions

Error 3034, Illegal index value - the destination matrix is smaller than the source matrix.

Example

```
INT GLOBAL SOURCE(3)(3), DESTINATION(3)(3)    !Define source and destination
                                              !matrices
!-------------------- Assign values to SOURCE matrix ---------------------
SOURCE(0)(0)=1;SOURCE(0)(1)=2;SOURCE(0)(2)=3;SOURCE(1)(0)=4;SOURCE(1)(1)=5;
SOURCE(1)(2)=6;SOURCE(2)(0)=7;SOURCE(2)(1)=8;SOURCE(2)(2)=9
!------------------ Assign values to DESTINATION matrix --------------------
COPY(SOURCE,DESTINATION,0,1,0,2,1,2,0,2)      !COPY command -
```

```
                                        !The program copies the first
                                        !two rows from the source
                                        !matrix to the destination
                                        !matrix second two rows. The
                                        !matrices are as follows:
                                        !Source:
                                        !1 2 3
                                        !4 5 6
                                        !7 8 9
                                        !Destination:
                                        !0 0 0
                                        !1 2 3
                                        !4 5 6
STOP                                    !Ends program
```

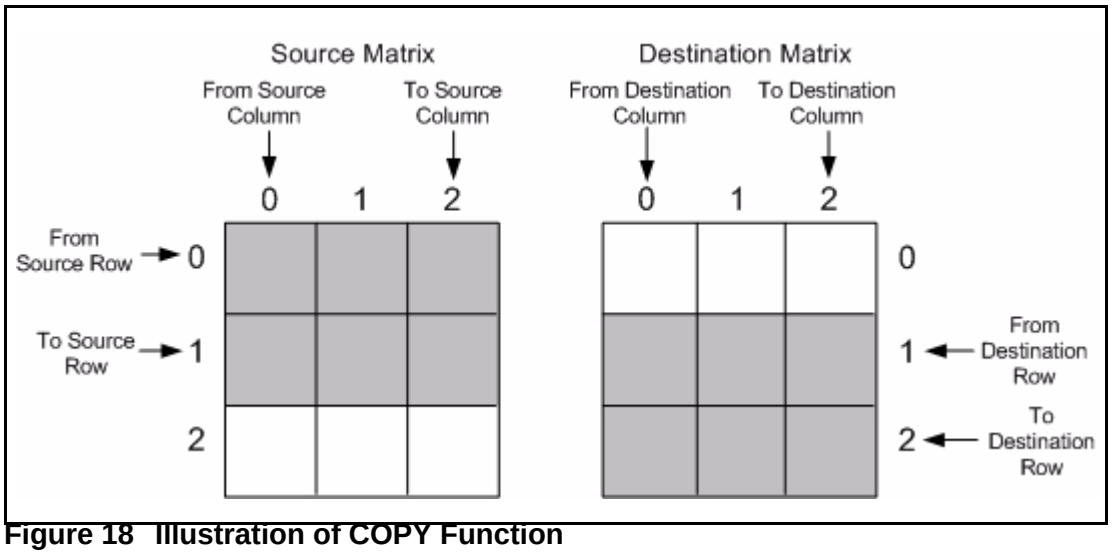


Figure 18 Illustration of COPY Function

4.2.3 FILL

Description

FILL fills an array or a section of array with a specified value.

Syntax

FILL (*real value*, *destination*, *from_array_row*, *to_array_row*, *from_array_column*, *to_array_ column*)

Arguments

real value	Any real number.
destination	The name of an array that has been declared in the program to which the value is to be copied.
from_array_row	The index of the first row of the destination array to begin filling.

to_array_row	The index of the last row of the destination array to end filling.
from_array_column	The index of the first column of the destination array to begin filling. Used only for matrix type arrays, otherwise it can be ommitted.
to_array_ column	The index of the last column of the destination array to end filling. Used only for matrix type arrays, otherwise it can be ommitted.

Comments
If the matrix indexes are omitted, the entire matrix will be filled with the requested value.

Example

```
GLOBAL ARR(6)(3)

FILL(3,ARR,0,4,1,2)

STOP


!The program fills ARR with the value 3, from row 0 to row 4,
!and from column 1 to column 2. After executing the program, the
!resulting ARR is:


!0          3          3
!0          3          3
!0          3          3
!0          3          3
!0          3          3
!0          0          0
```

4.2.4 MAX

Description
MAX finds the maximal value in an array or in a section of an array.

Syntax
MAX(*array_name*)

Arguments

array_name	The name of an array that has been declared in the program.

Return Value
Real number - returns the maximal element in the array.

Example

```
REAL Ar(3)
Ar(0) = 1; AR(1)=0.5; Ar(2)=3;
DISP MAX(Ar) !Output = 3
```

4.2.5 MAXI

Description

MAXI finds the maximal value in an array or in a section of array and returns its index.

Syntax

MAXI(*array_name*)

Arguments

array_name	The name of an array that has been declared in the program.

Return Value

Integer - **MAXI** returns the index of maximal element in the array, or in the specified section of the array. In case of a two-dimensional array only the column index is returned.

Error Conditions

None

Example

```
REAL AR(3)
AR(0)= 1; AR(1)= 0.5; AR(2)= 3
DISP MAXI(AR) !Output = 2
```

4.2.6 MIN

Description

MIN finds the minimal value in an array or in a section of an array.

Syntax

MIN(*array_name*)

Arguments

array_name	The name of an array that has been declared in the program.

Return Value

Real number - returns the minimal element in the array.

Error Conditions

None

Example

```
REAL Ar(3)

Ar(0) = 1; AR(1)=0.5; Ar(2)=3;

DISP MIN(Ar) !Output = 0.5
```

4.2.7 MINI

Description

MINI finds the element with minimal value in an array or in a section of an array and returns its index.

Syntax

MINI(*array_name*)

Arguments

array_name	The name of an array that has been declared in the program.

Return Value

Integer - **MINI** returns the index of the minimal element in array X, or in the specified section of array x. In case of a two-dimensional array, only the column index is returned.

Example

```
REAL AR(3)

AR(0)= 1; AR(1)= 0.5; AR(2)= 3;

DISP MINI(AR) !Output = 1
```

4.3 Miscellaneous Functions

The Miscellaneous functions are:

Function	Action
GETCONF	Reads hardware and firmware parameters.
SYSINFO	Returns certain system information based on the argument that is specified.
GETVAR	Reads the current value of the variable and returns it as a real number.
SETCONF	Writes hardware and firmware parameters.
SETVAR	Provide write access to all ACSPL+ variables and to user variables declared with *tag*
STR	Converts an integer array to a string.
STRTONUM	Converts an ASCII string representation of a number to the number it represents.
NUMTOSTR	Converts a number to an ASCII string.
BCOPY	Copies bytes from a source array to a target array.

4.3.1 GETCONF

Advanced	***GETCONF** should be used by advanced users only.*

Description
GETCONF retrieves system configuration data that was configured by **SETCONF**.

Note	*Some keys relate to data that is set by the system and not by* **SETCONF**, *for keys set by* **SETCONF** *see* **Table 49**.

Syntax
GETCONF(*key*,*index*)

Arguments

key	Specifies the configured feature.
index	Specifies the axis or buffer.

Return Value

GETCONF return values are described in **Table 46** according to key.

Table 46 GETCONF Return Values (page 1 of 2)

Key	Value - Bit and Explanation
26	Returns the mask that determines, for each digital input, whether the leading or trailing signal edge triggers an interrupt. The mask contains a bit for each available input signal. The location of bits in the mask corresponds to the location of bits in variable **IN0**. For each bit - 1: The controller generates an interrupt on the falling edge of the corresponding input signal. 0: Controller generates an interrupt on the rising edge of the corresponding input signal. After power-up, all bits in the mask = 0.
29	Returns 0 or 2 according to the current assignment of the specified output. 2: Brake - output supplies the brake control signal 0: Output follows the corresponding bit of **OUT0** As set by **SETCONF.**
37	Returns the mask that determine whether a digital input triggers on a single edge, or on both edges. If value = 0, the trigger edge is determined by **key** 26. The location of bits in the mask corresponds to the location of bits in variable **IN0**. 1: The controller generates an interrupt on both edges. 0: The controller generates an interrupt on one edge. After power-up the mask contains 0 in all bits.
71	Used to view the actual assignment of digital outputs to PEG states and PEG pulses outputs. The fuction returns the bit code according to **Table 9**.
72	Used to view the actual encoder PEG engine assignment. Returns the bit code according to **Table 7**.
73	Used to view the actual output pins assignment for PEG engines. Returns the bit code according to **Table 11** and **Table 12**.
203	Returns the value of **MFLAGS.1** (Open Loop) 1: Open loop 0: Not open loop
204	Returns the value of **MFLAGS.9** (Commutation OK). 1: Commutation OK) 0: Commutation not OK)
214	Only valid for brushless motors (**MFLAGS.8** = 1). Returns the commutation phase (degrees) at the current point.
215	Only valid for brushless motors (**MFLAGS.8** = 1) where the encoder has encountered the index and **IND** contains a valid value. Returns commutation phase (degrees) at the point of the last index stored in the **IND** variable.

Table 46 GETCONF Return Values (page 2 of 2)

Key	Value - Bit and Explanation
216	Only valid for brushless motors (**MFLAGS.8** = 1). Returns the commutation state (**MFLAGS.9**): 0: Commutation is not OK (not initialized) 1: Commutation is OK.
229	Returns the mechanical brake output: 1: Mechanical brake is inactive 0: Mechanical brake is active
246	Upon receipt of a Drive Alarm signal, the controller stores a general Drive Alarm code (5019) in the **MERR** variable. The extended Drive Fault status code can be obtained by executing **GETCONF***(246, Axis)*. The following extended Drive Fault statuses are supported (the **MERR** code appears in brackets) by the DDM3U Motor Drive: ❑ Drive Alarm: Short circuit (5061) ❑ Drive Alarm: Power supply too high (5064) ❑ Drive Alarm: Temperature too high (5065) ❑ Drive Alarm: Power down (5069) ❑ Drive Alarm: Drive not ready (5071) ❑ Drive Alarm: Over current (5072) Using the **GETCONF** funtion, the faults 5064, 5065, 5069, 5071 can be read before the **ENABLE** command is executed. **Note** *The* **GETCONF** *function provides a delay until the extended Drive Fault is received. This behavior differs from the implementation in SPiiPlus PCI-DDM4 and SPiiPlus CM, where the extended Drive Fault status is stored in* **MERR** *immediately upon receipt of the Drive Alarm signal.*
247	Retrieves information about a failed component. The information is retrieved as 32-bit integer number that contains the following information about failed component: Error Code, Card Address and Card Type. In the event that more than one component has failed, Bit 15 is set to “1”.
262	Returns the current Hall state, which can be 0, 1, 2, 3, 4, or 5, of the axis given by axis_def (a number: 0, 1, 2, ... up to the number of axes in the system minus 1). It returns -1 for invalid states.
301	Returns the size of the segment queue in segmented motions (**MPTP...ENDS**, **MSEG...ENDS**, **PATH...ENDS**, **PVSPLINE...ENDS**).
310	Returns an integer value that contains the TCP/IP address currently assigned to the controller. The **index** argument has to be zero, for example, `GETCONF(310, 0)` If a TCP/IP protocol is not configured, or not supported, the return value is zero.

Examples

```
?B/GETCONF(229,0)
```

Returns the following mask:

00000000,00000000,00000000,00000001

Reports the actual state of the mechanical brake for the given axis. The output is presented in binary base (/B).

```
?X/GETCONF(229,0)
```

Output:

```
00000001
```

Reports the actual state of the output pins of the mechanical brake for the given axis in hexidecimal format.

```
?X/GETCONF(310,0)
```

Output:

```
6e00000a
```

The address of a controller whose TCP/IP address is 10.0.0.100

4.3.2 SYSINFO

Description

SYSINFO retrieves a value related to the SPiiPlus controller system based on the argument that is specified.

Syntax

SYSINFO(*Int*)

Arguments

Int	A positive integer ranging between 1 to 16.

Return Value

Returns a value based on the specified ***Int***.

The possible values of ***Int*** and the associated return values are given in **Table 47**:

Table 47 SYSINFO Return Values

Int	Value Returned
1	The SPiiPlus model number.
2	The SPiiPlus version number.
10	Number of regular ACSPL+ program buffers
11	D-buffer index.
13	Total number of axes (in current configuration).
14	Number of EtherCAT nodes
15	Number of data collection channels per DSP.
16	EtherCAT support: 1 - Yes 0 - No

4.3.3 GETVAR

Description
GETVAR retrieves a value from a variable (ACSPL+ or user-defined variable, scalar or array) that was declared as a Tag number.

Syntax
GETVAR(*Tag, [Index1, Index2]*)

Arguments

Tag	The variable Tag number (positive integer)
Index1, Index2	If the variable is an array, the indexes point to the specific location in the array. If the variable is scalar, omit the indexes.

Return Value
Returns the value of the variable specified by the ***Tag***.

Example

```
GLOBAL REAL TAG 1001 EE(2)(2)
                          !Defines user variable array EE as Tag 1001.
SETVAR (15,1001,1,1)      !Sets value 15 to user variable array
                          !described in Tag 1001 cell (1)(1).
DISP GETVAR (1001,1,1)    !Display the value in user variable array
                          !designated by Tag 1001 cell (1)(1).
STOP                      !Ends program
```

The controller displays the return value of the **GETVAR** function which is = 15.

4.3.4 SETCONF

Advanced	***SETCONF** should be used by advanced users only.*

Description
SETCONF defines system configuration data.

All the keys that can be set by **SETCONF** and are listed in **Table 49** can also be retrieved by **GETCONF**.

Syntax
SETCONF(*key,index,value*)

Arguments

key	Specifies the configured feature.
index	Specifies axis or buffer number.
value	The set of bit states for the defined **key**. **value** is set up according to a 16-bit binary template, illustrated in **Table 48**. The controller strips all leading zeros. The controller understands **value** in binary, hexidecimal, or decimal format. The following prefixes determine **value** format: 0B - binary 0H - hexidecimal A decimal **value** does not require any prefix.

Table 48 16-bit Binary *value* Template

Bit	31	30	29	28	27	...	6	5	4	3	2	1	0
Prefix [0B] | [0H] ***value***	0	0	0	0	0	...	0	0	0	0	0	0	0

SETCONF arguments are detailed in **Table 49**.

Table 49 SETCONF Arguments (page 1 of 6)

Key	Index	Value - Bit and Explanation
26	Don't Care	Sets the mask that determines, for each digital input, whether the leading or trailing signal edge triggers an interrupt. The mask contains a bit for each available input signal. The location of bits in the mask corresponds to the location of bits in variable **IN0**. For each bit - 1: The controller generates an interrupt on the falling edge of the corresponding input signal. 0: Controller generates an interrupt on the rising edge of the corresponding input signal. After power-up, all bits in the mask = 0.
29	A six digit number: AANNOO, where: ❑ AA - is the axis designation (00-99) ❑ NN - digital output index (00-99) ❑ OO - specific output (00-99)	The following decimal values assign the specified output bit to one of the following functions according to the ***value*** argument: 2: Output supplies the brake control signal 0: Cancel assignment for the axis, that is, output reverts to digital output

Table 49 SETCONF Arguments (page 2 of 6)

Key	Index	Value - Bit and Explanation
37	Don't Care	Sets the mask that determine whether a digital input triggers on a single edge, or on both edges. If **value** = 0, the trigger edge is determined by **key** 26. The location of bits in the mask corresponds to the location of bits in variable **IN0**. 1: The controller generates an interrupt on both edges. 0: The controller generates an interrupt on one edge. After power-up the mask contains 0 in all bits.
203	Axis: 0, 1, 2, 3, 4, 5, 6, 7	Sets the value of **MFLAGS.1** (Open Loop) 1: Open loop 0: Not open loop
204	Axis: 0, 1, 2, 3, 4, 5, 6, 7	Sets the value of **MFLAGS.9** (Commutation OK). 1: Commutation OK) 0: Commutation not OK)
214	Axis: 0, 1, 2, 3, 4, 5, 6, 7	Only valid for brushless motors (**MFLAGS.8** = 1). Sets the commutation offset at the current position to the specified value in degrees.
215	Axis: 0, 1, 2, 3, 4, 5, 6, 7	Only valid for brushless motors (**MFLAGS.8** = 1) where the encoder has encountered the index and **IND** contains a valid value. Sets the commutation offset so that the commutation phase at the last index point is equal to the specified value in degrees.
216	Axis: 0, 1, 2, 3, 4, 5, 6, 7	Only valid for brushless motors (**MFLAGS.8** = 1). If **value** = 1: ❑ **MFLAGS.9**=0 (Commutation is not OK) ❑ **MFLAGS** bits **1**, **4**, **5**, **6** are set to zero ❑ **MFLAGS.8** is set to 1 ❑ **DCOM** is reset to zero ❑ Sets **RPOS** = **FPOS** If **value** = 0: ❑ **MFLAGS.9**=0 (Commutation is not OK) ❑ **MFLAGS** bits **1**, **4**, **5**, **6** are set to zero ❑ **MFLAGS.8** is set to 10 ❑ Variable **DCOM** is reset to zero.

Table 49 SETCONF Arguments (page 3 of 6)

Key	Index	Value - Bit and Explanation
217	Axis: 0, 1, 2, 3, 4, 5, 6, 7	Only valid for brushless motors (**MFLAGS.8** = 1) where the encoder has encountered the index and **IND** contains a valid value. Combines actions of **SETCONF** key 215, and **SETCONF** key 216. **value**: Not Relevant. ❑ Adjusts the commutation offset so that the commutation phase at the last index point is equal to the specified value in degrees. ❑ **MFLAGS.9**=0 (Commutation is not OK) ❑ **MFLAGS** bits **1**, **4**, **5**, **6** are set to zero ❑ **MFLAGS.8** is set to 1 ❑ **DCOM** is reset to zero ❑ Sets **RPOS** = **FPOS**
229	Axis: 0, 1, 2, 3, 4, 5, 6, 7	Mechanical brake output: 1: Deactivates mechanical brake output. 0: Activates mechanical brake output. The motor must be disabled to execute this setting.
230	Axis: 0, 1, 2, 3, 4, 5, 6, 7	Safety system configuration - 1: Enable safety system configuration 0: Disable safety system configuration After power-up, the safety system configuration is disabled. The user must enable the safety system configuration for a motor before executing any **SETCONF** function with from **key** 231 to **key** 236. An **index** designation of -1 selects all axes.
231	Axis: 0, 1, 2, 3, 4, 5, 6, 7	Sets variables **SAFINI** and **S_SAFINI** that control inversion of safety inputs. Only the following bits that correspond to safety inputs are meaningful: ❑ 0 (#LL) - Left Limit ❑ 1 (#RL) - Right Limit ❑ 9 (#DRIVE) - Drive Alarm ❑ 28 (#ES) - Emergency Stop where: 1: Defines inversion of the input signal 0: Defines no inversion of the input signal An **index** designation of -1 selects all axes.
232	Axis: 0, 1, 2, 3, 4, 5, 6, 7	Sets variables **FMASK** and **S_FMASK** that enable fault detection. For each fault there is a corresponding bit in **value** word. The bit value controls the fault detection as follows: 1: Enables the fault detection 0: Disables the fault detection An **index** designation of -1 selects all axes.

Table 49 SETCONF Arguments (page 4 of 6)

Key	Index	Value - Bit and Explanation
233	Axis: 0, 1, 2, 3, 4, 5, 6, 7	Sets variables **FDEF** and **S_FDEF** that enable default controller response to faults. For each fault there is a corresponding bit in **value** word. The bit value controls the response to the corresponding fault as follows: 1: Enable the default controller response for the fault 0: Disable the default controller response for the fault An **index** designation of -1 selects all axes.
234	Axis: 0, 1, 2, 3, 4, 5, 6, 7	Sets an internal mask that determines if the axis defined by **index** responds to a fault that occurs on any other motor. Only motor fault bits are meaningful in the **value** word since the response to a system fault always involves all motors. 1: The motor responds to a fault detected for any motor. 0: The motor responds to a fault detected only for the motor defined by **index**. An **index** designation of -1 selects all axes.
235	Axis: 0, 1, 2, 3, 4, 5, 6, 7	Sets an internal mask for the controller response to **KILL** - 1: Response to **KILL** is enabled 0: Response to **KILL** is disabled If a response to both **KILL** and **DISABLE** is enabled, the controller first performs **KILL** and only then disables the motor. An **index** designation of -1 selects all axes.
236	Axis: 0, 1, 2, 3, 4, 5, 6, 7	Sets an internal mask for the controller response to **DISABLE** - 1: Response to **DISABLE** is enabled 0: Response to **DISABLE** is disabled If a response to both **KILL** and **DISABLE** is enabled, the controller first performs **KILL** and only then disables the motor. An **index** designation of -1 selects all axes.
246	Axis: 0, 1, 2, 3, 4, 5, 6, 7	**SETCONF***(246, Axis, 0)* is used to clear the fault status on all axes that relate to the DDM3U Motor Drive that handles the specified axis. *Note:* *The* **FCLEAR** *command does not clear fault status in the MC4U-CS,* **SETCONF**(246, Axis, 0) *has to be used instead.*

Table 49 SETCONF Arguments (page 5 of 6)

Key	Index	Value - Bit and Explanation
302	2 1	The following decimal values specify a communication channel for special input: 3: Set the channel to **MODBUS** Master mode. 2: Set the channel to **MODBUS** Slave mode 1: Assigns the channel for special input. 0: Set the channel to regular command processing mode (default channel mode). If a channel is assigned for special input, the controller does not process commands from this channel. Output to the channel is provided by regular **DISP** and **SEND** commands. **index** specifies the channel.
303	2 1	The **value** sets the baud rate for the specified serial channel, where the baud rate is the decimal value. 115200 (default), 57600, 19200, 9600, 4800, 2400, 1200, 600, 300. **index** specifies the channel.
304	2 1	Sets communication options: Bit 2 - 1: extended stop bit 0: normal stop bit Bit 3 - 1: check parity 0: no parity Bit 4 - 1: even parity 0: odd parity **index** specifies the channel.
308	19 18 17	When the controller acts as a **MODBUS** master, the function establishes or closes the **MODBUS** TCP connection with the Slave device using the specified slaveIP. Up to three slaves can be connected to the master controller (17, 18 or 19). Value=SlaveIP. The slaveIP value specifies the **MODBUS** Slave device IP address. The slaveIP address is calculated as follows: If the Slave has the following address: 192.168.1.10, the slaveIP parameter should be 10*2^24 + 1*2^16 + 168*2^8 + 192 = 167880896 (0x0A01A8C0). If the specified channel is already open, the function closes the opened channel and then opens new one. If slaveIP is zero, the function closes the TCP connection.
309	Don't Care	Defines the sequence for the two 16-bit **MODBUS** interface registers - 1: Low word will be first, then high word 0: High word will be first, then low word (default configuration)

Table 49 SETCONF Arguments (page 6 of 6)

Key	Index	Value - Bit and Explanation
310	0	Used for providing access to the controller TCP/IP address. Where: Value = TCP/IP Address, which is a 32-bit (four bytes) integer, each byte of which contains one part of the TCP/IP address, for example, 0x6400000a assigns address 10.0.0.100 (the number is read from right to left and fills in the address left to right). If **value** is zero, **SETCONF** activates a new execution of the DHCP protocol and obtains a new TCP/IP address from the host (the host may configure the same address as before). **SETCONF** does not change the **TCPIP** variable. After power-up, the controller is initialized with the TCP/IP address set in the **TCPIP** variable. There are several limitations when using **SETCONF**(310): ❑ If the **TCPIP** variable stored in the flash is zero, **SETCONF**(310) must be used only with zero address argument. In other words, if the controller is configured for dynamic addressing, assigning static addresses is not allowed. ❑ If the **TCPIP** variable stored in the flash is not zero, **SETCONF**(310) must be used only with non-zero address arguments. In other words, if the controller is configured for static addressing, switching to a dynamic address is not allowed. ❑ **SETCONF**(310) has a long execution time. During this time, communication with the controller is impossible using any communication channel. Use **SETCONF**(310) only within the controller initialization sequence. Avoid attempts to communicate with the controller and the motor **ENABLE** command or motion commands while **SETCONF**(310) is in progress.

Return Value
None

Error Conditions
None

Example
The following example illustrates setting the mechanical brake of axis 4 as digital output 1 bit 1:

```
SETCONF(29, 040101,2)
```

4.3.5 SETVAR

Description
SETVAR writes a value to an ACSPL+ or user variable, scalar or array, that was declared as a Tag number.

Syntax
SETVAR(*value, Tag, [Index1, Index2]*)

Arguments

value	A real or integer value assigned to the variable.
Tag	The variable Tag number (positive integer).
Index1, Index2	If the variable is an array, the indexes point to the specific location in the array. If the variable is **scalar**, omit the indexes.

Return Value
None

Error Conditions
None

Example

```
GLOBAL REAL TAG 1001 EE(2)(2)   !Defines user variable array EE as Tag 1001
SETVAR (15,1001,1,1)            !Sets value 15 to user variable array
                                !described in Tag 1001 cell (1)(1).
DISP GETVAR (1001,1,1)          !Retrieves the value in user variable array
                                !described in Tag 1001 cell (1)(1).
STOP                            !Ends program
```

The controller displays the **GETVAR** return value which is = 15.

4.3.6 STR

Description
STR converts an integer array to a string. Each element of the array is interpreted as an ASCII character.

Syntax
string **STR**(*array_name,[start_index,] [number]*)

Arguments

array_name	Name of user-defined integer array.

start_index	Index in the array from which to start converting.
number	Number of characters to convert.

Comments
If an element value is in the range from 0 to 255, it is directly converted to the corresponding ASCII character. Otherwise, the value will be cyclic, based on 256.

If **start_index** is omitted, the assignment starts from the first element of the array.

If neither**start_index** nor **number** is specified, the conversion takes all elements of the array. If only **start_index** is specified, the conversion takes all characters from the specified index until the end of the array. **number** limits the number of characters in the resulting string.

Return Value
String composed of the array elements interpreted as characters.

Example

```
GLOBAL INT BIBI(3)
BIBI(0)=65
BIBI(1)=67
BIBI(2)=83
DISP STR(BIBI)
STOP
```

The function transforms each of the array members of BIBI to ASCII code characters. When **DISP** is applied, the displayed value will be: “ACS”

4.3.7 STRTONUM

Description
This function converts ASCII encoded element string to a number.

Syntax
double **STRTONUM**(Source, Type, From, N)

Arguments

Source	An integer array
Type	Designates the format of the converted number, it can be: ❑ 0 - decimal ❑ 1- hex ❑ 2 - floating point
From	Index of the first element in the array that may contain the number
N	The number of elements in source that may be used

Return Value
The number converted from the ASCII string

Example

```
real number
int string(4);
number = 0
string(0) = 49; !1
string(1) = 50; !2
string(2) = 46; !.
string(3) = 50; !2
number = strtonum(string, 0, 0, 4);
disp"number = %f", number;
number = strtonum(string, 1, 0, 4);
disp"number = %f", number;
number = strtonum(string, 2, 0, 4);
disp"number = %f", number;
stop
! output:
! number = 12.000000
! number = 18.000000
! number = 12.200000
```

4.3.8 NUMTOSTR

Description
This function converts a number to a string of elements where each element is an ASCII encoded value.

Syntax
int **NUMTOSTR**(Number, Target, Type, From, N)

Arguments

Number	The number to be converted
Target	An integer array used in which to put the results
Type	Designates the format of the number to be converted, it can be: ❑ 0 - decimal ❑ 1- hex ❑ 2 - floating point
From	Index of the first element in the array that may contain the number
N	The number of elements in target that may be used

Return Value
The number of elements used.

Example

```
real number
int target(12);
int i;
int j;
number = 12.2
i=0; loop 12 target(i) = 0; i = i+1; end;
j = numtostr(number,target,0,0,12)
disp "target = %X,%X,%X,%X,%X num elements = %d",
target(0),target(1),target(2),target(3),target(4), j
i=0; loop 12 target(i) = 0; i = i+1; end;
j = numtostr(number,target,1,0,12)
disp "target = %X,%X,%X,%X,%X num elements = %d",
target(0),target(1),target(2),target(3),target(4), j
i=0; loop 12 target(i) = 0; i = i+1; end;
j = numtostr(number,target,2,0,12)
disp "target = %X,%X,%X,%X,%X num elements = %d",
target(0),target(1),target(2),target(3),target(4), j
stop
! output:
! target = 31,32,0,0,0 num elements = 2
! target = 63,0,0,0,0 num elements = 1
! target = 31,32,2E,32,30 num elements = 9
```

4.3.9 BCOPY

Description

This function can be used to copy bytes from the source array to the target array.

Syntax

int **BCOPY**(Source_array, Target_array, CopyBytes, S_SkipBytes, T_SkipBytes, From, N)

Arguments

Source_array	Array containing input data
Target_array	Array containing output data
CopyBytes	Number specifying how many bytes will each copy operation
S_SkipBytes	Number specifying how many bytes will be skipped in the source array following each copy operation
T_SkipBytes	Number specifying how many bytes will be skipped in the target array following each copy operation
From	Index of the first element in the source array that may be used
N	The number of elements in the source that may be used

Return Value

The number of elements (in the source) used.

Comments

The function copies **CopyBytes** from the source array to the target array, skip **S_SkipBytes** in the source and skip **T_SkipBytes** in the target, and then copy **CopyBytes** again. The operation starts from the **From** element in source and is applied to a maximum of **N** elements. This means no bytes from any element other than the specified **N** elements will be affected. Elements less than **N** can be affected if the target array is too short to complete the whole operation.

The difference between elements and bytes throughout this function should be noted.

Examples

In the examples that follow use is made of:

bcopy(source,target,1,3,0) - Copy every 4 byte element from source to a 1 byte element in target

bcopy(source,target,2,2,0) - Copy every 4 byte element from source to a 2 byte element in target

bcopy(source,target,1,0,3) - Copy every 1 byte element from source to a 4 byte element in target

bcopy(source,target,2,0,2) - Copy every 2 byte element from source to a 4 byte element in target

1. Unpacking a single element in source to 1, 2, or 4 elements in target

```
int source(1)
int target(4)
int j
source(0) = 0x01020304;
j = bcopy(source,target,4,0,0,0,1) => target = 1020304,0,0,0 num elements = 1
j = bcopy(source,target,2,0,2,0,1) => target = 304,102,0,0 num elements = 1
j = bcopy(source,target,1,0,3,0,1) => target = 4,3,2,1 num elements = 1

stop
```

2. Packing multiple elements in source to 1 element in target.

```
int source(4)
int target(1)
int i
int j
target(0) = 0;
i = 0
loop 4
source(i) = 1;
i=i+1;
end
j = bcopy(source,target,4,0,0,0,4) => target(0) = 1 j = 1
j = bcopy(source,target,2,2,0,0,4) => target(0) = 10001 j = 2
j = bcopy(source,target,1,3,0,0,4) => target(0) = 1010101 j  = 4
stop
```

4.3.10 ECIN

Description
This function is used to copy theEtherCAT network input variable at the corresponding EtherCAT offset into the specified ACSPL+ variable.

Syntax
ECIN(int offset, Varname)

Arguments

offset	Internal EtherCAT offset of network variable derived from Communication Terminal **ETHERCAT** command.
Varname	Valid name of ACSPL+ variable, global or standard.

Return Value
None

Comments
Once the function is called successfully, the Firmware copies the value of the network input variable at the corresponding EtherCAT offset into the specified ACSPL+ variable, every controller cycle.

There is no restriction on number of mapped network variables.

Note	
	The mapping is allowed only when stack is operational.

In the event of wrong parameters or stack state, the function will produce corresponding runtime error.

Example
```
ecin(26,V0)
```

This maps the network variable at offset 26 to the global real variable: **V0**.

4.3.11 ECOUT

Description
This function is used to copy the value of ACSPL+ variable into the network output variable at the corresponding EtherCAT offset.

Syntax
ECOUT(int offset, Varname)

Arguments

offset	Internal EtherCAT offset of network variable derived from Communication Terminal **ETHERCAT** command.
Varname	Valid name of ACSPL+ variable, global or standard.

Return Value
None

Comments
Once the function is called successfully, the Firmware copies the value of the specified ACSPL+ variable network input variable into the given EtherCAT offset , every controller cycle.

There is no restriction on number of mapped network variables.

Note	*The mapping is allowed only when stack is operational.*

In the event of wrong parameters or stack state, the function will produce corresponding runtime error.

Example
```
ecout(98,V1)
```

This maps the network variable at offset 98 to the global real variable: **V1**.

4.3.12 ECUNMAP

Description
This function is used to reset all previous mapping defined by **ECIN** and **ECOUT**.

Syntax
ECUNMAP

Arguments
None

Return Value
None

Comments
The mapping is allowed only when stack is operational.

4.4 CoE Functions

CoE functions are required for SDO transfers in CoE. SDO are part of the cyclic EtherCAT data transfer. It is impossible to define a generic function for any kind of mailbox transfer, as protocols like EoE, FoE and VoE have their own definitions. So CoE is supported first.

Note	*The **Communication Terminal** **ETHERCAT** command reports for every slave if it has Mailbox support.*

The CoE functions are:

Function	Action
COEREAD	Read CoE slave Object Directory entry.
COEWRITE	Write into CoE slave Object Directory.

4.4.1 COEREAD

Description
This function is used to read Object Dictionary entry from CoE slave.

Syntax
COEREAD[/size] (int slave,int Index,int Subindex)

Arguments

size	1/2/4 number of bytes in the Object Dictionary or /f for floating.
slave	Slave number (which can be obtained by running the **ETHERCAT** command).
Index	Index in the Object Dictionary.
Subindex	Sub-index in the Object Dictionary.

Return Value
Returns the value stored in the variable in the specified **Index** of the Object Dictionary.

Comments
In case of wrong parameters, the corresponding runtime error will be generated. The function cannot be used in the **Communication Terminal**. The function delays the buffer execution on its line until it is successful or fails the whole buffer with timeout or other error.

Example
```
coeread/4 (0,0x6040,0)
```

This reads 4 bytes from **slave** 0, at **Index** 0x6040, **Subindex** 0 and returns the value that is stored in the variable at this **Index**.

4.4.2 COEWRITE

Description
This function is used to write a value into the CoE slave Object Dictionary.

Syntax
COEWRITE[/size] (int slave,int Index,int Subindex,double Value)

Arguments

size	1, 2 or 4 - the number of bytes in the Object Dictionary or /f for floating.
slave	Slave number (which can be obtained by running the **ETHERCAT** command)
Index	Index in the Object Dictionary.
Subindex	Sub-index in the Object Dictionary.
Value	The value to be written.

Return Value
None

Comments

In case of wrong parameters, the corresponding runtime error will be generated. The function cannot be used in the **Communication Terminal**. The function delays the buffer execution on its line until it is successful or fails the whole buffer with timeout or other error.

Example

```
coewrite/4 (0,0x6041,0,0x0)
```

This writes the **Value** 0 (4 bytes) into **slave** 0, at **Index** 0x6041, **Subindex** 0.

4.5 Servo Processor Functions

SPii™, a proprietary ACS Motion Control Servo Processor (SP), executes the real-time tasks such as implementation of the real time control algorithms. Each SPii can control from two to eight axes (depending on the product). The SPii includes all the necessary peripherals that are needed for a high performance axis control, such as encoder counters, Digital-to-Analog interface, smart inputs and outputs.

Servo Processor functions are used to read and monitor SP values.

Model	
	The quantity of Servo Processors is model-dependent. Check the controller Hardware Guide for the specific SPiiPlus model.

The Servo Processor functions are:

Function	Action
GETSP	Reads a value from the specified SP address
GETSPA	Retrieves address of the SP variable specified by name.
GETSPV	Reads a value from the specified SP variable name
MONSP	Initiates monitoring of the specified SP address through analog output
MONSPV	Initiates monitoring of the specified SP variable name through analog output
SETSP	Sets a value for the specified SP address
SETSPV	Sets a value for the specified SP variable name

4.5.1 GETSP

Description

GETSP reads a value from the specified SP address.

The read value is treated as a signed integer number.

The function returns immediately if the address lies in an area accessible directly from MPU, otherwise the function is delayed by one MPU cycle.

Syntax

int **GETSP**(*int SP, int Address*)

Arguments

SP	The SP number in the controller. Legal values range from zero to 3, depending on the controller model.
Address	Address within the SP. Look up addresses by enering the command: **#SI** in the **Communication Terminal**.

Note	
	For **Address** *it is recommended using the memory address as obtained by* **GETSPA**.

Return Value

Integer - The current value read from the specified SP address.

Error Conditions

The function causes an error if an SP number is specified other than 0–3, or if an illegal address is specified.

Example

```
REAL PAR_ADDRESS
PAR_ADDRESS=GETSPA(0,"PE(0)")  ! Get the memory address of PE(0) in SP#0
GETSP(0, PAR_ADDRESS)          ! The return value is the PE(0) in SP#0
```

4.5.2 GETSPA

Description
GETSPA retrieves the SP address of the specified SP variable.

Syntax
int **GETSPA**(*SP_number, "SP_variable"*)

Arguments

SP_number	The SP number in the controller, an integer number from 0 to 3.
SP_variable	String representing the name of an SP variable. In order to avoid unexpected results, place **SP_variable** in quotes.

Return Value
Address of the variable in the SP memory, or -1 if the variable does not exist. The return value can be used in **GETSP**, **SETSP** and **MONSP**.

Example

```
XX= GETSPA(0,"0_V_FF")     !Retrieve the address from SP 0 (axes 0 and 2)
                           !of SP variable X_V_FF, velocity feed forward,
                           !of axis 0.
                           !XX=2
```

4.5.3 GETSPV

Note	
	*This function is obsolete and is replaced by **GETSP**.*

4.5.4 MONSP

Description
MONSP presents the value of an SP variable, specified by its address. The value is presented through **AOUT**, and can be observed with an external device such as a scope.

The **SP** variable can be observed as a voltage level on an external scope. The value can be the ratio between the maximum voltage level and SP variable voltage level. Where the maximum voltage level number is 16 bit (2^{15}=32768)

The digital value of **AOUT** is a special monitoring code that doesn't fall between the limits of ±32768 (16 bits). The next assignment to the corresponding **AOUT** element overwrites the monitoring code and the analog output returns to displaying the **AOUT** value.

Syntax
MONSP(*int AOUT, int SP_number, int Address, int Shift*)

Arguments

AOUT	The analog output to which the SP variable content specified by its address, will be sent. For each SP there are dedicated analog outputs as follows: **AOUT** **SP** *1, 3* *0* *5, 7* *1* *9, 11* *2* *13, 15* *3*
SP_number	The SP number in the controller, an integer number from 0 to 3.
Address	Address within the SP. Look up addresses by enering the command: **#SI** in the **Communication Terminal**.
Shift	Scaling shift of the monitored value. A positive number specifies left shift, a negative number specifies right shift. See **Figure 19**, **Illustration of Shift**..

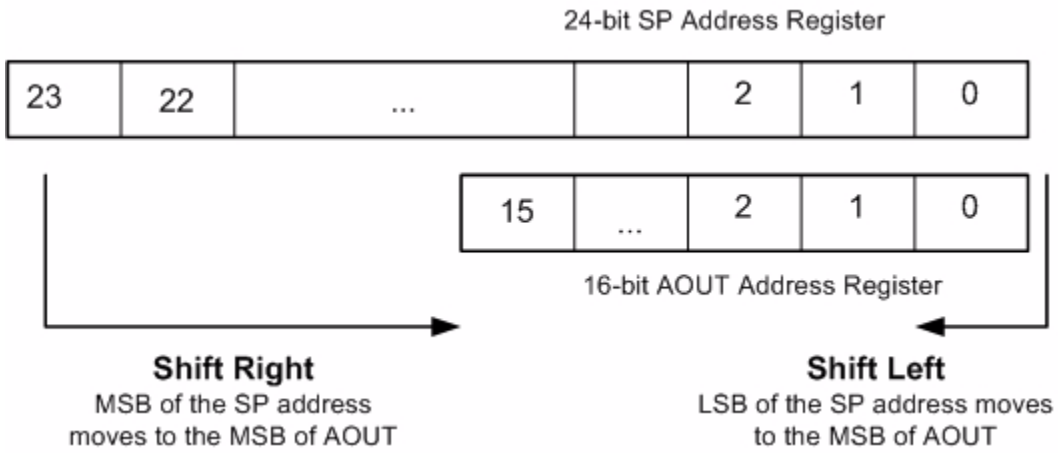


Figure 19 Illustration of Shift

Comments

An address from a specific SP can be monitored only through an analog output belonging to the same SP. This rule applies limitations to possible values of arguments **AOUT** and **SP**. Two addresses from one SP can be monitored simultaneously.

The function overwrites the value stored in the corresponding ACSPL+ **AOUT** variable with a special monitoring code. Once **SP** receives the monitoring code, it starts transferring the value from the specified address to the analog output. **SP** also provides the required scaling shift of the monitored variable.

The next assignment to the corresponding **AOUT** element overwrites the monitoring code and the analog output returns to displaying the **AOUT** value.

Error Conditions
3149 - Illegal SP address

3126 - Illegal SP number.

3138 - Incompatible SP and analog output

Example

```
MONSP(1,0,3,0)          !The function sends the contents of the SP variable
                        !that exists in address 3 in SP number 0 to AOUT1.
                        !There are no cell shifts required. The output of
                        !AOUT1 on an external scope will be 3.1V.
                        !The SP variable in address 3 is A_V_FF, and its
                        !value is 19,660. The maximum AOUT voltage is 5V
                        !(between AOUT1 and ground) and the resolution is
                        !16 bit, hence 32768. To obtain the SP A_V_FF value
                        !from the voltage level - 32768*(3/5)= 19660.
```

4.5.5 MONSPV

Description
MONSPV presents the value of an SP variable, specified by its name. The value is presented through **AOUT**, and can be observed with an external device such as a scope.

The **SP** argument can be observed as a voltage level on an external scope. The value can be the ratio between the maximum voltage level and SP variable voltage level. Where the maximum voltage level number is 16 bit (2^{15}=32768)

The digital value of **AOUT** is a special monitoring code that doesn't fall between the limits of ±32768 (16 Bit). The next assignment to the corresponding **AOUT** element overwrites the monitoring code and the analog output returns to displaying the **AOUT** value.

Syntax
MONSPV(*int AOUT, SP_number, int Shift*)

Arguments

AOUT	The analog output to which the SP variable content specified by its address, will be sent. For each SP there are dedicated analog outputs as follows: **AOUT** **SP** *1, 3* *0* *5, 7* *1* *9, 11* *2* *13, 15* *3*
SP_number	The SP number in the controller, an integer number from 0 to 3.
Shift	Scaling shift of the monitored value. A positive number specifies left shift, a negative number specifies right shift. See **Figure 19**, **Illustration of Shift**..

Error Conditions

3149 - Illegal SP address

3126 - Illegal SP number.

3138 - Incompatible SP and analog output

Example

```
MONSPV(3,SP0:A_V_FF,0)   !The function sends the contents of the SP variable
                         !X_V_FF that exists in SP number 0 to AOUT1.
                         !There are no cell shifts required. The output of
                         !AOUT1 on an external scope will be 3.1V.
                         !The SP variable SP0:A_V_FF value is 19,660. The
                         !maximum AOUT voltage is 5V (between AOUT1 and
                         !ground) and the resolution is 16 bit, hence 32768.
                         !To obtain the SP A_V_FF value from the voltage
                         !level - 32768*(3/5)= 19660.
```

4.5.6 SETSP

Caution	***The* SETSP *function is for advanced users only. Misuse of this function may damage the controller.***

Description
SETSP writes a value to the specified SP address.

Syntax
real **SETSP**(*int SP_number, int Address, int value*)

Arguments

SP_number	The SP number in the controller, an integer number from 0 to 3.
Address	Address within the SP. Look up addresses by enering the command: **#SI** in the **Communication Terminal**.
value	The value to write to the address.

Error Conditions
Error 3126, Illegal SP number.

Example

```
REAL PAR_ADDRESS
PAR_ADDRESS=GETSPA(0,"X_DZ_MIN")  !Gets the memory address of PE(0) in SP #0
SETSP(0, PAR_ADDRESS, 1000)       !Changes the value of the DZ_MIN(0) in
                                  !SPii#0 to 1000.
```

4.5.7 SETSPV

Note	
	*This function is obsolete and is replaced by **SETSP**.*

4.6 Signal Processing Functions

The Signal Processing functions are:

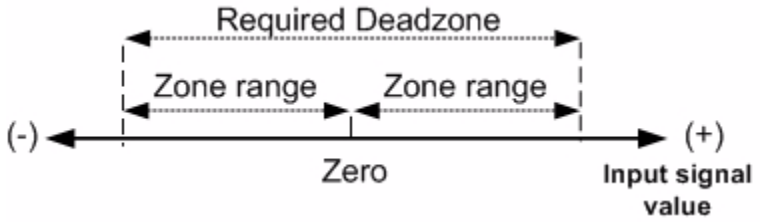
Function	Action
COPY	The function copies one array to another.
DEADZONE	Implements dead-zone routine
DSIGN	Implements a dynamic version of the standard **SIGN** function.
DSTR	Converts a string to an integer array.
EDGE	Returns 1 on positive edge of x

Function	Action
INP	Reads data characters from the specified channel and stores them into integer array
INTGR	Implements an integrator with **DEADZONE** and **SAT**.
LAG	Provides delayed switching on argument change (anti-bouncing effect)
MAP	Implements a table-defined function with constant step
MAPB	Implements a one-dimensional uniform B-spline interpolation
MAPN	One-dimensional non-uniform linear interpolation (replaces obsolete **mapby1** and **mapby2**)
MAPS	One-dimensional non-uniform linear interpolation (replaces obsolete **mapby1** and **mapby2**)
MAPNB	One-dimensional non-uniform b-spline
MAPNS	One-dimensional non-uniform Catmull-Rom spline
MAPS	One-dimensional uniform Catmull-Rom spline
MAP2	Implements a table-defined function with two arguments and constant step along each argument.
MAP2B	Two-dimensional uniform b-spline
MAP2N	Two-dimensional non-uniform linear interpolation (replaces obsolete **map2free**)
MAP2NB	Two-dimensional non-uniform b-spline
MAP2NS	Two-dimensional non-uniform Catmull-Rom spline
MAP2S	Two-dimensional uniform Catmull-Rom spline
MATCH	Calculates axis position that matches current reference position of the same axis with zero offset.
RAND	Implements a random number generator.
ROLL	Calculates a result rolled-over to within one pitch.
SAT	Implements a saturation characteristic

4.6.1 DEADZONE

Description

DEADZONE returns values based on a defined analog or other input signal with a defined symmetrical or asymmetrical dead zone around zero.



DEADZONE is useful for anti-bouncing effect.

Syntax
DEADZONE (*input_signal, zone1[,zone2]*)

Arguments

input_signal	Any real number or integer expression
zone1	Any real number or integer expression for a symmetrical deadzone. See example.
zone2	Any real number or integer expression, required for an asymmetrical deadzone requires two values. See example.

Return Value
DEADZONE returns a **real number** as follows:

If - *zone* < *input_signal* < *zone*, return value = 0

If *input_signal* < -*zone*, return value = *input_signal* + *zone*

If: *input_signal* > *zone*, return value = *input_signal* - *zone*

Error Conditions
A negative zone range returns Error 3045, Numerical Error in Standard Function.

Examples
Example 1:

Symmetrical Dead Zone

In this example, illustrated in **Figure 20**, input_signal ranges from -20: +20, and creates a symmetrical dead zone from -10: +10.

```
return_value = DEADZONE(input_signal,10)
```

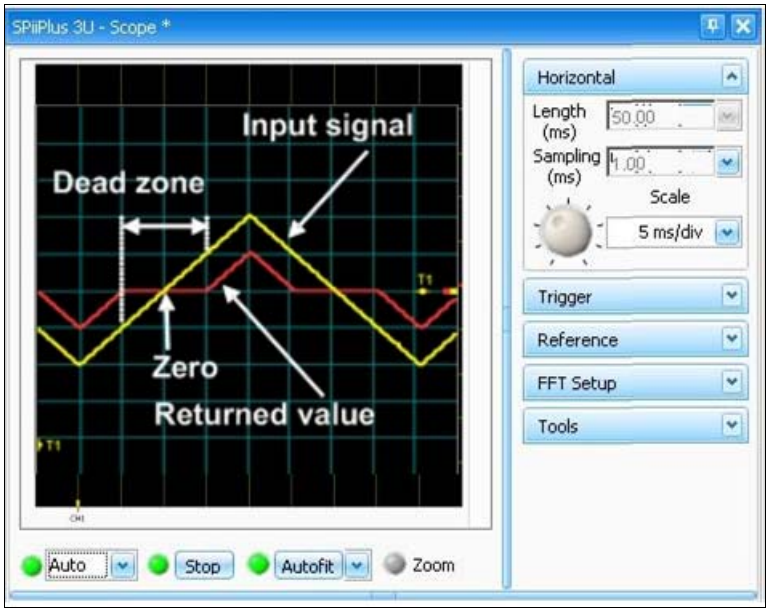


Figure 20 Symmetrical Dead Zone Example

Example 2:

Asymmetrical Dead Zone

In this example, illustrated in **Figure 21**, input_signal ranges from -20: +20, and creates an asymmetrical dead zone from -5 – +15.

```
return_value = DEADZONE(input_signal,-5,10)
```

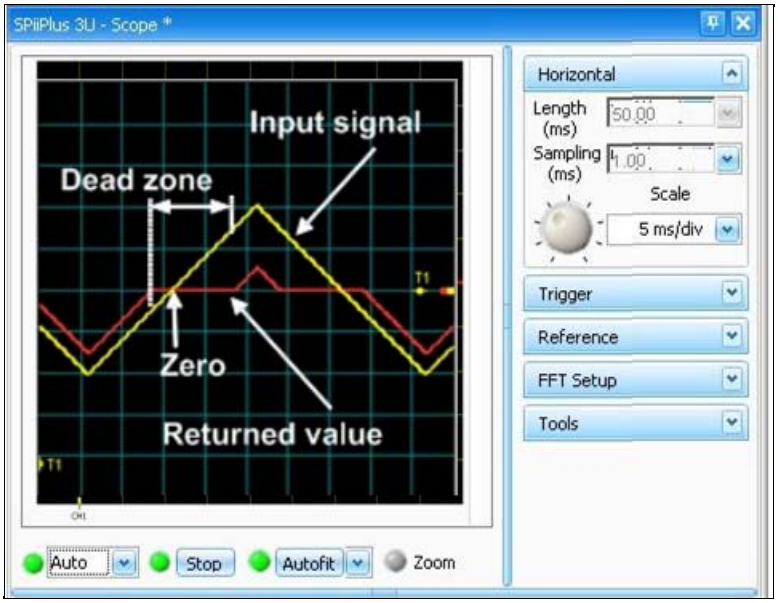


Figure 21 Asymmetrical Dead Zone Example

4.6.2 DSIGN

Description

DSIGN returns values between -1 to 1 based on a defined input variable with defined delay time and ramp time.

Syntax

real **DSIGN**(*X, Delay_Time, Ramp_Time*)

Arguments

X	Input variable name declared as real or real expression.
Delay_Time	Real number that determines on zero crossing of X, continues the previous return value for defined delay time in msec.
Ramp_Time	Real number that controls the rate of the returned value change between -1 to 1 (and vice versa), defined in msec.

Return Values

0: when the input variable is < zero while *Ramp Time* = 0.

1: when the input variable is > zero while *Ramp Time*= 0.

-1 to 1: when the input variable changes between positive to negative (or vice versa), while *Ramp_Time* is > 0.

Error Conditions
Delay time and ramp time should be positive. If any of them becomes negative, Error 3045, Numerical Error in Standard Function appears.

Example

```
REAL XX,YY
YY = DSIGN(XX,50,100)
STOP
```

Figure 22 illustrates the **DSIGN** function.

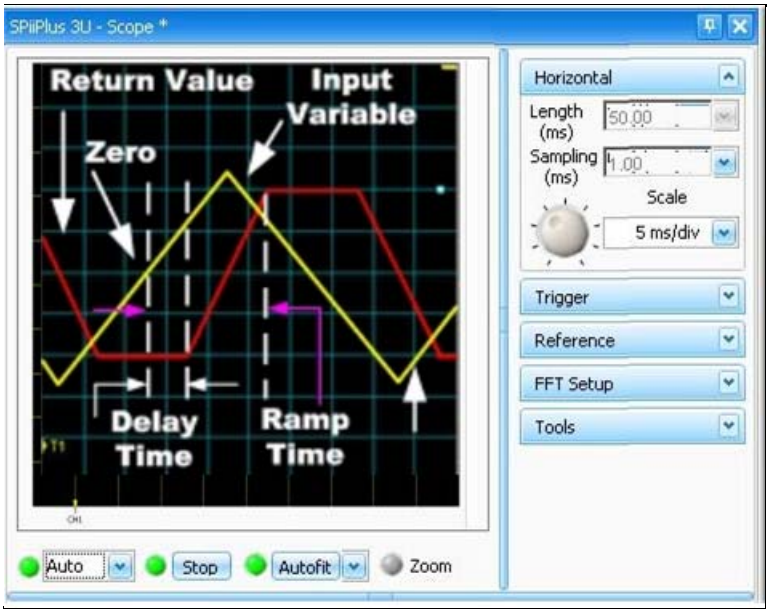


Figure 22 DSIGN Function Example

4.6.3 DSTR

Description
DSTR converts a string to an integer array based on an ASCII transformation code. The function decomposes a string to characters and assigns the characters to the sequential elements of the variable array.

Each ASCII character is represented as its numerical value and stored in a separate element of the array.

Syntax
int **DSTR**(*string, array_name, [start_index,] [number]*)

Arguments

string	String of characters enclosed in double quotation marks.
array_name	User-defined integer array.
start_index	Index in the array from where the transformed numbers will be placed. If ***start_index*** is omitted, the assignment starts from the first element of the array.
number	Number of characters from the string to be transformed. If ***number*** is omitted, the function assigns all characters of the string. If ***number*** is specified, the function assigns the specified number of characters. In both cases the assignment stops when the last array element is reached.

Return Value

The number of actually transformed characters.

Example

In the example below **DSTR** transforms each of "ACS-Motion_Control" characters to its corresponding numeric value and assigns then to a user defined array "BIBI" (each character to a separate array member. BIBI content is as follows:

65 67 83 45 84 69 67 72 56 48

```
GLOBAL INT BIBI(10)
DSTR("ACS-Motion_Control",BIBI)
STOP
```

4.6.4 EDGE

Description

EDGE returns 0 or 1, based on a defined input variable. **EDGE** is mainly useful in PLC implementation when an action must be taken once a condition becomes true.

Syntax

real **EDGE**(*X*)

Arguments

X	Input real variable name or real expression.

Return Value

EDGE returns 1 when the input variable changes from 0 to (+1) or from 0 to (-1) until the input variable changes to other values. The function returns 0 in all other cases.

Error Conditions

None

Example

```
GLOBAL REAL XX,XI,YY          !Declares three local variables
XX=-5 ; XI=1                  !Assigns values to the variables XX and XI.
WHILE 1                       !Run the following routine forever
XX=XX+XI
YY = EDGE(XX)
IF (XX>5) | (XX<-5) XI=-XI    !Keeps XX between -5 to 5.
END                           !Ends IF
END                           !Ends WHILE
STOP                          !Ends program
```

Figure 23 illustrates the **EDGE** function.

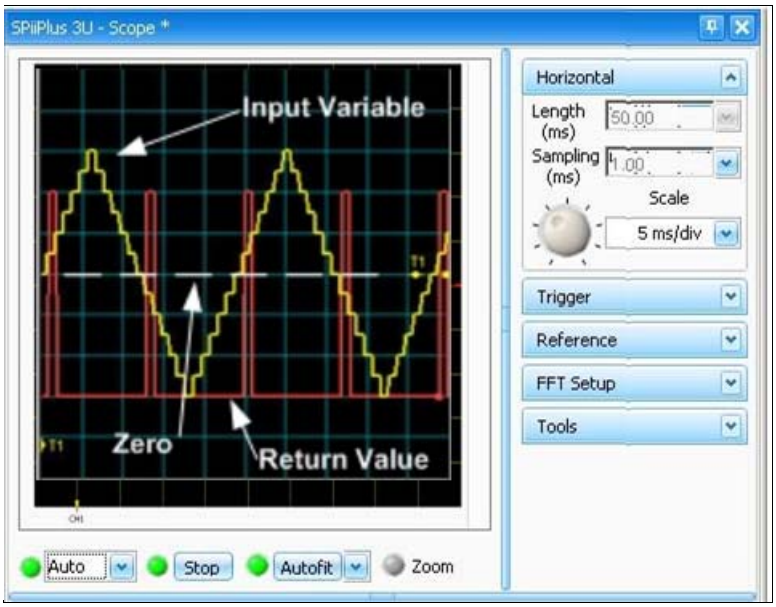


Figure 23 EDGE Function Example

4.6.5 INP

Description

INP reads data values from a specified channel and stores them to an integer array. This function is useful for creating an interface between the controller and special input devices such as a track-ball, mouse or various sensors. **INP** is also used when the controller acts as a master with the **MODBUS** protocol communication.

Before using **INP**, configure the relevant communication channel as a special communication channel using **SETCONF** function, key 302.

See also **OUTP**.

Syntax

int **INP**(*channel, [array,] [start_index,] [number,] [timeout]*)

Arguments

channel	Communication channel index: ❑ 1 - serial communication channel COM1. ❑ 2 - serial communication channel COM2. ❑ 6 - Ethernet network (TCP) ❑ 7 - Ethernet network (TCP) ❑ 8 - Ethernet network (TCP) ❑ 9 - Ethernet network (TCP) ❑ 10 - Ethernet Point-to-Point (UDP) ❑ 12 - PCI bus ❑ 16 - communication channel with **MODBUS** slave ❑ 36 - Ethernet network (TCP) ❑ 37 - Ethernet network (TCP) ❑ 38 - Ethernet network (TCP) ❑ 39 - Ethernet network (TCP)
array	User-defined integer array to which the data will be stored. If **array** is omitted, the function purges formerly received characters in the channel.
start_index	The first received character is assigned to the array element with the specified index. If **start_index** is omitted, the assignment starts from the first element of the array.
number	The number of characters to be collected to the variable array. If **number** is omitted, the function continues receiving characters until the last element of the array is assigned, or the carriage return character is received.
timeout	The function waits for input not more than the specified number of milliseconds. If **timeout** is omitted, the waiting time is not limited.

Return Value

The number of entities that have been stored into the variable.

Error Conditions

None

Example

```
GLOBAL INT MMM(10)       !Defines global user array MMM with ten elements.
SETCONF(302,2,1)         !Assigns COM2 as special input.
INP(2)                   !Purges the input buffer from old values
INP(2,MMM,0,10,1000)     !INP(2)- purge the input buffer from old values
                         !INP(2,MMM,0,10,1000) - store values from COM2 port
                         !to MMM user variable, from array index 0, total of
                         !10 values. The data collection process will end
                         !within 1000 msec or when the 10 values have been
```

```
                        !collected.
STOP                    !Ends program
```

4.6.6 INTGR

Description

INTGR returns an integrator with optional deadzone and saturation limits.

Syntax

real **INTGR**(*X, Deadzone, Min, Max*[,*Initial_Value*])

Arguments

X	Real variable name or real expression.
Deadzone	A real number. When the **Deadzone** value falls into the range of **-Deadzone** to **+Deadzone**, **INTGR** retains its previous value as if X = 0.
Min	A real number representing the low integrator saturation limit.
Max	A real number representing the high integrator saturation limit.
Initial_Value	A real number setting the initial value of the return value. (optional)

Return Value

INTGR returns a value in the range from ***Min*** to ***Max***.

Error Conditions

None

Example

Input variable XX is changing between -20 to +20.

```
GLOBAL REAL YY, XX              !Defines global real variables.
REAL ZZ                         !Defines increment variable as real
MARK:                           !Set GOTO line
ZZ=0.1; XX=-20                  !Set variable values
WHILE XX<=20                    !Set upper limit for XX
XX=XX+ZZ                        !Set increment for XX
YY=INTGR(XX,5,-10,10,20)
END                             !End WHILE
WHILE XX>=-20                   !Set lower limit for XX
XX=XX-ZZ                        !Set increment for XX
YY=INTGR(XX,5,-10,10,20)
END                             !End WHILE
GOTO MARK                       !GOTO MARK to create a program loop
STOP                            !End program
```

Figure 24 illustrates the **INTGR** function.

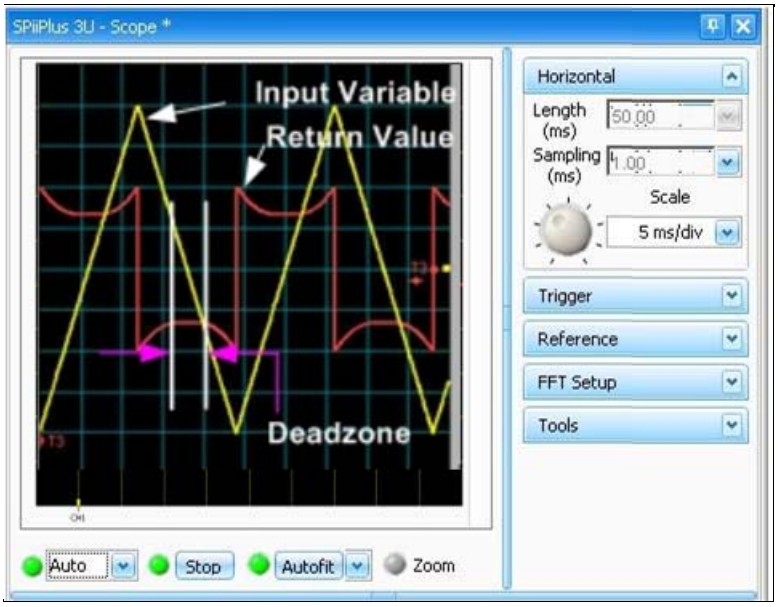


Figure 24 INTGR Function Example

4.6.7 LAG

Description

LAG returns values based on a defined input signal with defined up delay or down delay. This function is useful for an anti-bouncing effect.

Syntax

LAG(*X*, *up_delay*, *down_delay*)

Arguments

X	Real variable name or real expression. Although **LAG** accepts any real or integer argument for **X**, the typical use implies a logical **X** argument, for example, an integer variable or expression that supplies only 0 or 1. If **X** supplies values other than 0 or 1, **LAG** treats any non-zero value as 1.
up_delay	A real number representing the delay on the positive edge in msec.
down_delay	A real number representing the delay on the negative edge in msec.

Return Value

1: when **X** remains non-zero for at least **up_delay** msec.
0: when **X** remains zero for at least **down_delay** msec.

Error Conditions

None

Example

The example here demonstrates **LAG** by generating a reference signal XX that changes from 0 to 1 every 100 msec, and based on XX and YY, implements a lag of 50msec and 20msec.

```
INT XX,YY,XT                          !Define three integers
XT=TIME; XX=0                         !XT equals the time elapsed from
                                      !startup. XX equal zero.
WHILE 1                               !Run an infinite routine
IF TIME-XT>100; XX=^XX; XT=TIME; END  !Switch the value of XX - from ON to
                                      !OFF
YY=LAG(XX,20,50)                      !YY is a returned value that is delayed
                                      !20msec after the transfer of XX from
                                      !0 to 1 and delayed 50msec after the
                                      !transfer of XX from 1 to 0
END                                   !Ends WHILE
STOP                                  !Ends program
```

Figure 25 illustrates the **LAG** function.

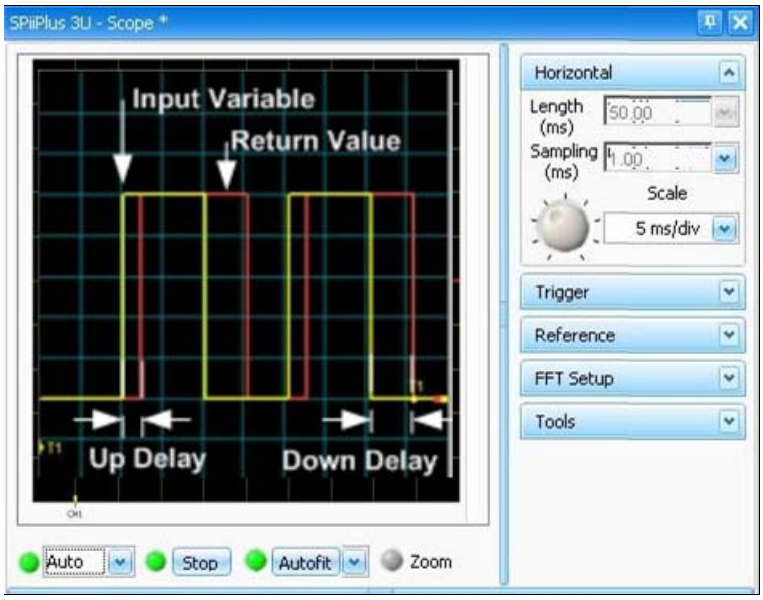


Figure 25 LAG Function Example

4.6.8 MAP

Description

MAP returns a value from an array of points per input variable value, base value of the variable and fixed defined intervals of the variable. The return values between the array points are linearly interpolated. **MAP** is useful for creating a correction table for mechanical error compensation.

Syntax

MAP(*X, array, base, step*)

Arguments

X	Real variable name or real expression.
array	The name of a real one-dimensional array that specifies points
base	A real number representing the value of **XX** that corresponds to the first point in array.
step	A real number representing the value of **XX** that defines fixed intervals between the array points.

Return Value

MAP returns a linearly interpolated value from the **array** for each value of **XX**.

Error Conditions

The function detects the following error conditions:

- Error 2044, Index is out of range, when the defined **array** size is less than the defined number of **array** points.
- ❑ Error 3113, The step in the table is zero or negative, when the **step** argument is zero or negative.

Example

```
REAL ARRAY(5);REAL XX; REAL YY; REAL XI
                        !Defines array ARRAY, and variables XX, YY,and XI.
!-------------------- Assign values to ARRAY array ---------------------
ARRAY(0)= -10; ARRAY(1)=3; ARRAY(2)=5; ARRAY(3)=14; ARRAY(4)=12
                        !Assign values to ARRAY
XX=-20; XI=0.1          !Assign values to XX and XI
WHILE XX<20             !Set conditions
XX=XX+XI
YY=MAP(XX,ARRAY,-10,5)  !MAP command where:
                        !XX is a variable name
                        !ARRAY is a one-dimensional point array
                        !-10 is the value of XX that corresponds to base
                        !point
```

```
                        !5 is the value of XX that defines fixed intervals
                        !between the ARRAY points.
END                     !Ends mapping
STOP                    !Ends program
```

Table 50 shows the XX and YY values.

Table 50 MAP Array

XX	-20	-10	-7	-5	0	5	10	15	20
ARRAY	N/A	-10	N/A	3	5	14	12	N/A	N/A
YY	-10	-10	-2.2	3	5	14	12	12	12

Figure 26 illustrates this **MAP** example on the Scope.

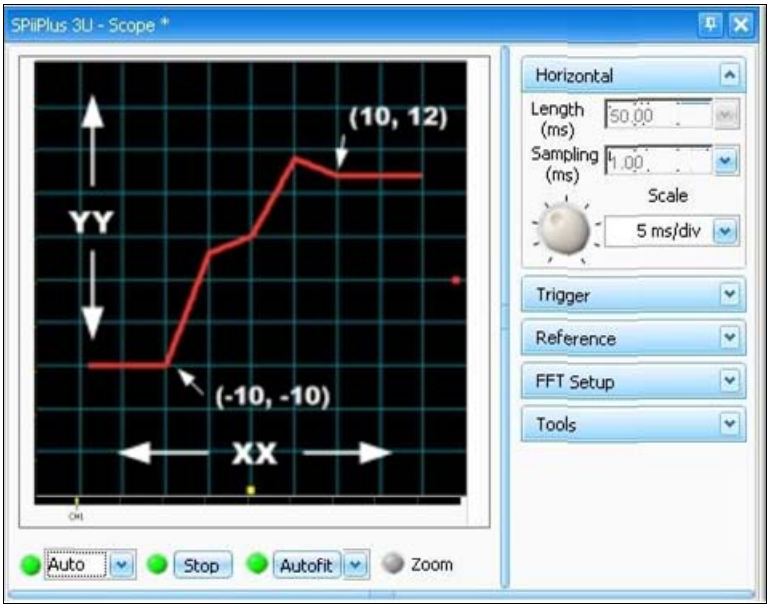


Figure 26 MAP Example on the Scope

4.6.9 MAPB

Description

MAPB returns a value from an array of points according to an input variable value, base value of the variable and fixed defined intervals of the variable. The return values between the array points are interpolated by a third order B-spline. **MAPB** is useful for creating a correction table for mechanical error compensation.

Syntax

MAPB(*XX, array, base, step*)

Arguments

XX	Real variable name or real expression.
array	The name of a real one-dimensional array that specifies points
base	A real number representing the value of **XX** that corresponds to the first point in **array**.
step	A real number representing the value of **XX** that defines fixed intervals between the **array** points.

Return Value

MAPB returns the third order B spline interpolated value from the **array** according to the value of variable **XX**.

Error Conditions

The function detects the following error conditions:

- Error 2044, Index is out of range, when the defined **array** size is less than the defined number of **array** points.
- Error 3113, The step in the table is zero or negative, when the **step** argument is zero or negative.

Example

```
REAL ARRAY(5);REAL XX; REAL YY; REAL XI
                        !Defines array ARRAY, and variables XX,YY, and XI
!------------------ Assign point values to ARRAY array ---------------------
ARRAY(0)= -10; ARRAY(1)=3; ARRAY(2)=5; ARRAY(3)=14; ARRAY(4)=12
XX=-20; XI=0.1          !Assign values to XX and XI
WHILE XX<20             !Set conditions
XX=XX+XI
YY=MAPB(XX,ARRAY,-10,5) !MAPB command where:
                        !XX is a variable name
                        !ARRAY is a one-dimensional point array
                        !-10 is the value of XX that corresponds to base
                        !point
```

```
                        !5 is the value of XX that defines fixed intervals
                        !between the ARRAY points.
END                     !Ends mapping
STOP                    !Ends program
```

Table 51 shows the XX and YY values.

Table 51 MAPB Array

XX	-20	-10	-7	-5	0	5	10	15	20
ARRAY	N/A	-10	N/A	3	5	14	12	N/A	N/A
YY	-10	-7.83	-2.45	1.16	6.16	12.33	12.33	12	12

Figure 27 illustrates this **MAPB** example on the Scope.

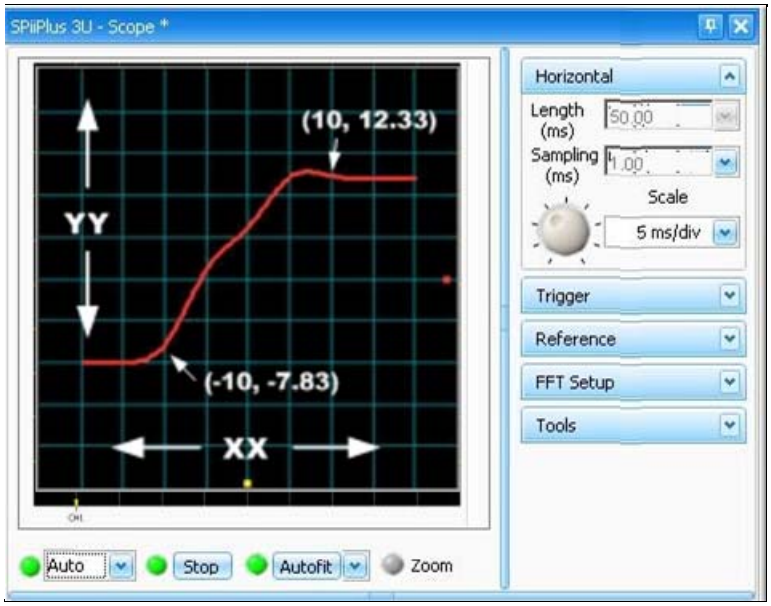


Figure 27 MAPB Example on the Scope

4.6.10 MAPN

Description

MAPN returns a value from an array of points according to the value of an input variable, and a look-up array. The return values between the array points are linearly interpolated. **MAPN** is useful for creating a correction table for mechanical error compensation.

Syntax

MAPN(*XX, X_table, Y_table*)

Arguments

XX	Real variable name or real expression.
X_table	The name of a real one-dimensional array that specifies the values of **XX** that correspond to the point array.
Y_table	The name of a real one-dimensional array that specifies points.

Return Value

MAPN returns a linearly interpolated value from the array according to the value variable **XX**.

Error Conditions

The function detects the following error conditions:

- The **Xtable** or **Ytable** argument have the wrong dimension
- The arguments in **Xtable** are not arranged in ascending order.

Example

```
REAL Xtable(5);REAL Ytable(5);REAL XX; REAL YY; REAL XI
                          !Defines arrays Xtable, Ytable, and variables XX,
                          !YY, and XI.
!-------------------- Assign values to Xtable array ---------------------
Xtable(0)= -10;Xtable(1)=-3;Xtable(2)=6;Xtable(3)=12;Xtable(4)=14
!-------------------- Assign values to Ytable array ---------------------
Ytable(0)= -10;Ytable(1)=3;Ytable(2)=5;Ytable(3)=14;Ytable(4)=4
XX=-20; XI=0.1            !Assigns values to XX and XI.
WHILE XX<20               !Set conditions.
XX=XX+XI
YY=MAPN(XX,Xtable,Ytable) !MAPN where:
                          !XX is the defined variable.
                          !Xtable is a one-dimensional point array that
                          !specifies the values of XX corresponding to the
                          !point array.
                          !Ytable is a one-dimensional array that specifies
                          !points.
```

```
END                             !Ends mapping
STOP                            !Ends program
```

Table 52 shows the XX and YY values.

Table 52 MAPN Array

XX	-20	-10	-7	-3	6	12	14	15	20
ARRAY	N/A	-10	N/A	-3	6	12	12	N/A	N/A
YY	-10	-10	-4.42	-3	6	12	14	14	14

Figure 28 illustrates this **MAPN** example on the Scope.

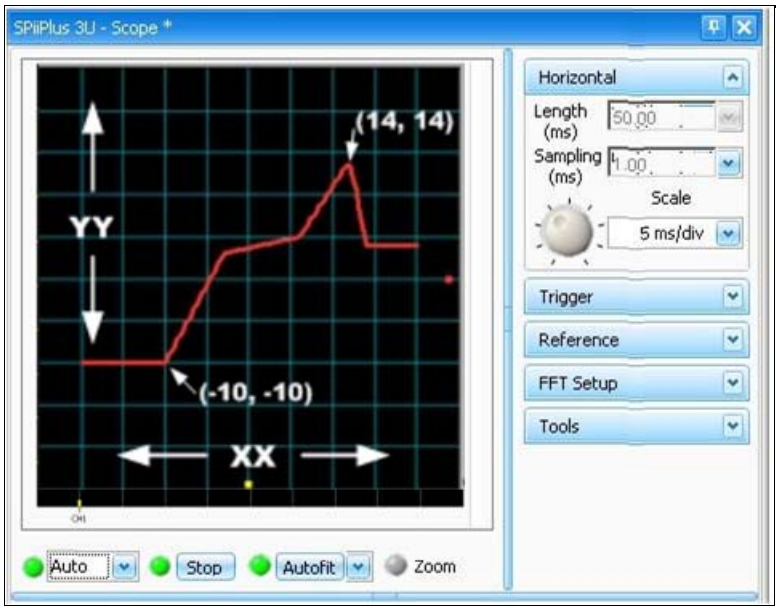


Figure 28 MAPN Example on the Scope

4.6.11 MAPNB

Description

MAPNB returns a value from a pre-defined control point array based on a defined variable and a look-up table. The return values between the table intervals are interpolated according to a third order B-spline. **MAPNB** is useful for creating a correction table for mechanical error compensation.

Syntax

MAPNB(*XX, X_table, Y_table*)

Arguments

XX	Real variable name or real expression.
X_table	The name of a real one-dimensional array that specifies the values of **XX** that correspond to the point array.
Y_table	The name of a real one-dimensional array that specifies points.

Return Value

MAPNB returns the third order B spline interpolated value from the **Y_table** per variable **XX** value

Error Conditions

The function detects the following error conditions:

- The **X_table** or **Y_table** argument have the wrong dimension
- The arguments in **X_table** are not arranged in ascending order sequence.

Example

```
REAL Xtable(5);REAL Ytable(5);REAL XX; REAL YY; REAL XI
                            !Defines arrays Xtable, Ytable, and variables XX,
                            !YY, and XI.
!-------------------- Assign values to Xtable array ---------------------
Xtable(0)= -10; Xtable(1)=-3; Xtable(2)=6; Xtable(3)=12; Xtable(4)=14
!-------------------- Assign values to Ytable array ---------------------
Ytable(0)= -10; Ytable(1)=3; Ytable(2)=5; Ytable(3)=14; Ytable(4)=4
XX=-20; XI=0.1              !Assigns values to XX and XI.
WHILE XX<20                 !Set conditions.
XX=XX+XI
YY=MAPNB(XX,Xtable,Ytable)  !MAPNB where:
                            !XX is the defined variable.
                            !Xtable is a point array that specifies the values
                            !of XX corresponding to the point array.
                            !Ytable is an array that specifies Y points.
```

```
END                             !Ends mapping
STOP                            !Ends program
```

Table 53 shows the XX and YY values.

Table 53 MAPNB Array

XX	-20	-10	-7	-3	6	12	14	15	20
ARRAY	N/A	-10	N/A	-3	6	12	12	N/A	N/A
YY	-10	-8.02	-4.65	-0.41	7.64	9.23	5	4.124	4

Figure 29 illustrates this **MAPNB** example on the Scope.

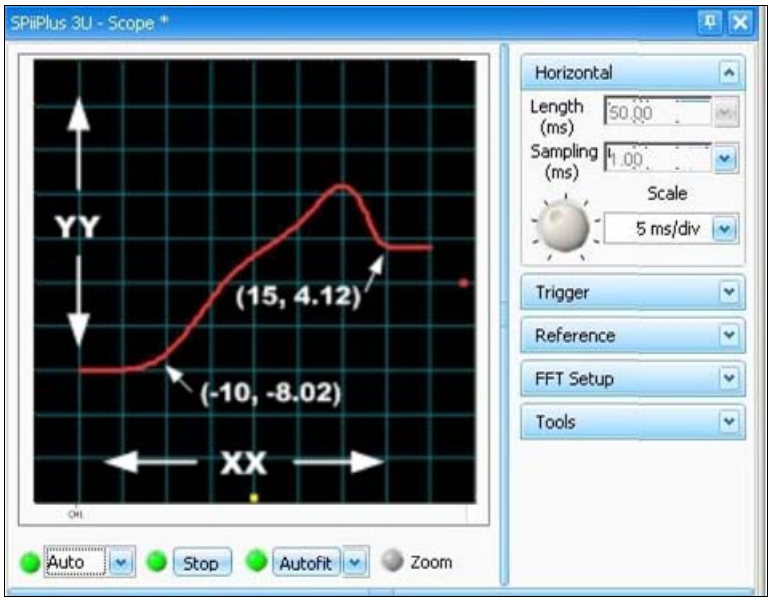


Figure 29 MAPNB Example on the Scope

4.6.12 MAPNS

Description

MAPNS returns a value from an array of points according to the value of an input variable, and a look-up array. The return values between the array points are interpolated according to a third order Catmull-Rom spline. **MAPNS** is useful for creating a correction table for mechanical error compensation.

Syntax

MAPNS(*XX, X_table, Y_table*)

Arguments

XX	Real variable name or real expression.
X_table	The name of a real one-dimensional array that specifies the values of **XX** that correspond to the point array.
Y_table	The name of a real one-dimensional array that specifies points.

Return Value

MAPNS returns the third order Catmull-Rom spline interpolated value from the **X_table** per variable **XX** value.

Error Conditions

The function detects the following error conditions:

- The **X_table** or **Y_table** argument have the wrong dimension
- The arguments in **X_table** are not arranged in ascending order sequence.

Example

```
REAL Xtable(5);REAL Ytable(5);REAL XX; REAL YY; REAL XI
                          !Defines arrays Xtable, Ytable, and variables XX,
                          !YY, and XI
!-------------------- Assign values to Xtable array ---------------------
Xtable(0)= -10; Xtable(1)=-3; Xtable(2)=6; Xtable(3)=12; Xtable(4)=14
!-------------------- Assign values to Ytable array ---------------------
Ytable(0)= -10; Ytable(1)=3; Ytable(2)=5; Ytable(3)=14; Ytable(4)=4
XX=-20; XI=0.1            !Assigns values to XX and XI.
WHILE XX<20               !Set conditions.
XX=XX+XI
YY=MAPNS(XX,Xtable,Ytable)  !MAPNS command where:
                          !XX is the defined variable.
                          !Xtable is a one-dimensional point array that
                          !specifies the values of XX corresponding to the
                          !point array.
                          !Ytable is a one-dimensional array that specifies
                          !points.
END                       !Ends mapping
STOP                      !Ends program
```

Table 54 shows the XX and YY values.

Table 54 MAPNS Array

XX	-20	-10	-7	-3	6	12	14	15	20
ARRAY	N/A	-10	N/A	-3	6	12	12	N/A	N/A
YY	-10	-10	-4.42	-3	6	12	14	14	14

Figure 30 illustrates this **MAPNS** example on the Scope.

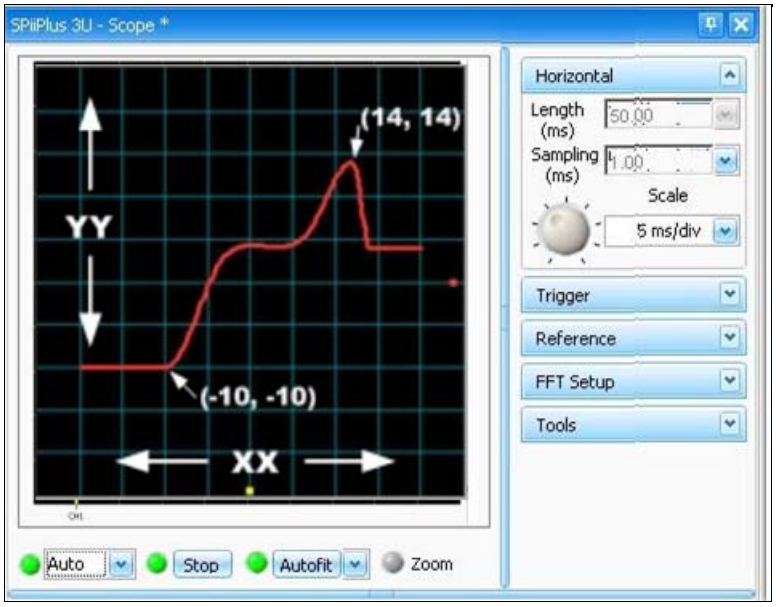


Figure 30 MAPNS Example on the Scope

4.6.13 MAPS

Description

MAPS returns a value from an array of points according to the value of an input variable, the base value of the variable**,** and the fixed defined intervals. The return values between the array-points are interpolated according to a third order Catmull-Rom spline. **MAPS** is useful for creating a correction table for mechanical error compensation.

Syntax

MAPS(*XX*, *array*, *base*, *step*)

Arguments

XX	Real variable name or real expression.
array	The name of a real one-dimensional array that specifies points

base	A real number representing the value of **XX** that corresponds to the first point in array.
step	A real number representing the value of **XX** that defines fixed intervals between the array points.

Return Value

MAPS returns the third order Catmull-Rom spline interpolated value from the array according to the value of **XX**.

Error Conditions

The function detects the following error conditions:

- Error 2044, Index is out of range, when the defined **array** size is less than the defined number of **array** points.
- Error 3113, The step in the table is zero or negative, when the **step** argument is zero or negative.

Example

```
REAL ARRAY(5);REAL XX; REAL YY; REAL XI
                          !Defines ARRAY a 5 element array, XX, YY, and XI
!-------------------- Assign values to ARRAY array --------------------
ARRAY(0)= -10; ARRAY(1)=3; ARRAY(2)=5; ARRAY(3)=14; ARRAY(4)=12
XX=-20; XI=0.1            !Assigns values to XX and XI.
WHILE XX<20               !Set conditions.
XX=XX+XI
YY=MAPS(XX,ARRAY,-10,5)   !MAPS command where:
                          !XX is the defined variable.
                          !ARRAY is a one-dimensional point array.
                          !-10 is the value of XX that corresponds to
                          !base point.
                          !5 is the value of XX that defines fixed
                          !intervals between the ARRAY points.
END                       !Ends mapping
STOP                      !Ends program
```

Table 55 shows the XX and YY values.

Table 55 MAPS Array

XX	-20	-10	-7	-5	0	5	10	15	20
ARRAY	N/A	-10	N/A	3	5	14	12	N/A	N/A
YY	-10	-10	-2.65	3	5	14	12	12	12

Figure 31 illustrates this **MAPS** example on the Scope.

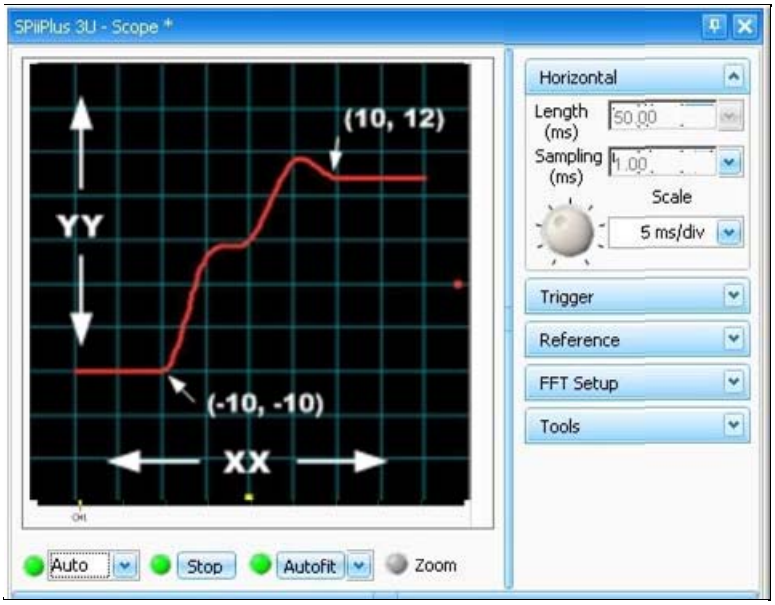


Figure 31 MAPS Example on the Scope

4.6.14 MAP2

Description

MAP2 returns a value from a two dimensional point array for each set of two input variable values, the base values of the input variables**,** and a fixed, defined interval for the variables. The return values between the array-points are linearly interpolated. **MAP2** is useful for dynamic error compensation.

Syntax

MAP2(*XX, ZZ, table, baseX, stepX, baseY, stepY*)

Arguments

XX	Real variable name or real expression.
ZZ	Real variable name or real expression.
table	The name of a real two-dimensional array that specifies points
baseX	A real number representing the value of **XX** that corresponds to the first point in the array.
stepX	A real number representing the value of **XX** that defines fixed intervals between the array points.
baseY	A real number representing the value of **ZZ** that corresponds to the first point in the array.
stepY	A real number representing the value of **ZZ** that defines fixed intervals between the array points.

Return Value

MAP2 returns a linearly interpolated value from the array according to the value of variables **XX** and **ZZ**.

Error Conditions

The function detects the following error conditions:

- Error 3072, Wrong array size, if **table** has only one dimension.
- Error 3113, The step in the table is zero or negative, when the **stepX** or **stepY** arguments are zero or negative.

Example

```
GLOBAL REAL XI,XX,YY,ZZ,TABLE(5)(5)
                        !Defines real variables and 5x5 matrix.
XX=-20;ZZ=10;XI=0.1     !Assigns values to the variable.
!-------------------- Assign values to TABLE matrix --------------------
TABLE(0)(0)=2; TABLE(0)(1)=22; TABLE(0)(2)=6; TABLE(0)(3)=8; TABLE(0)(4)=10;
TABLE(1)(0)=12; TABLE(1)(1)=-44; TABLE(1)(2)=16; TABLE(1)(3)=18;
TABLE(1)(4)=20; TABLE(2)(0)=22; TABLE(2)(1)=24; TABLE(2)(2)=26;
TABLE(2)(3)=68; TABLE(2)(4)=30; TABLE(3)(0)=12; TABLE(3)(1)=34;
TABLE(3)(2)=59; TABLE(3)(3)=-38; TABLE(3)(4)=40; TABLE(4)(0)=92;
TABLE(4)(1)=44; TABLE(4)(2)=46; TABLE(4)(3)=10; TABLE(4)(4)=90
WHILE ZZ<70
XX=XX+XI                !Set conditions
ZZ=ZZ+XI
YY=MAP2(XX,ZZ,TABLE,-10,5,20,10)
                        !MAP2 command where:
                        !XX and ZZ are variables.
                        !TABLE is a 2x2 matrix that specifies points.
                        !-10 is the first XX point in the matrix.
                        !5 defines the fixed intervals between the XX points.
                        !20 is the first ZZ point in the matrix.
                        !10 defines the fixed intervals between the ZZ
                        !points.
DISP XX,ZZ,YY           !Displays values of XX, ZZ and YY.
END                     !Ends MAP2.
STOP                    !Ends program
```

Table 56 shows the YY values as a function of XX and ZZ arguments.

Table 56 MAP2 Table

	XX							
ZZ		-30	-10	-5	0	5	10	50
	1	2	2	22	6	8	10	10
	20	2	2	22	6	8	10	10
	30	12	12	-44	16	18	20	20
	40	22	22	24	26	68	30	30
	50	12	12	34	59	-38	40	40
	60	92	92	44	46	10	90	90
	100	92	92	44	46	10	90	90

Key:

XX and ZZ values
Interpolated YY values outside the boundaries of array “Table”
“Table” array values

Figure 32 illustrates this **MAP2** example on the Scope.

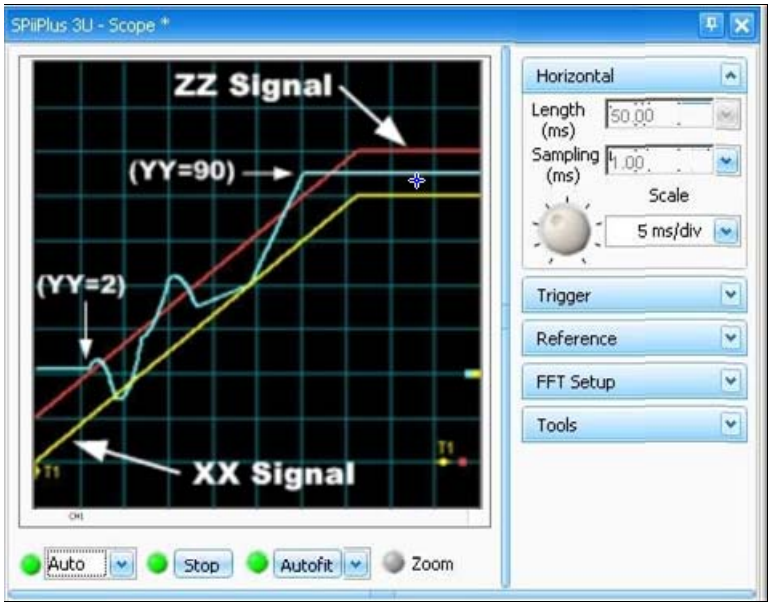


Figure 32 MAP2 Example on the Scope

4.6.15 MAP2B

Description

MAP2B returns a value from a two dimensional array of points according to the value of two input variables, where the base values of the input variables are fixed and the intervals of the input variables are defined. The return values between the array-points are interpolated according to a third order B-spline. **MAP2B** is useful for dynamic error compensation.

Syntax

MAP2B(*XX,ZZ, table, baseX, stepX, baseY, stepY*)

Arguments

XX	Real variable name or real expression.
ZZ	Real variable name or real expression.
table	The name of a real two-dimensional array that specifies points
baseX	A real number representing the value of **XX** that corresponds to the first point in the array.
stepX	A real number representing the value of **XX** that defines fixed intervals between the array points.
baseY	A real number representing the value of **ZZ** that corresponds to the first point in the array.
stepY	A real number representing the value of **ZZ** that defines fixed intervals between the array points.

Return Value

MAP2B returns the third order B-spline interpolated value from the array according to the value of variables **XX** and **ZZ**.

The function detects the following error conditions:

- Error 3072, Wrong array size, if **table** has only one dimension.
- Error 3113, The step in the table is zero or negative, when the **stepX** or **stepY** arguments are zero or negative.

Example

```
GLOBAL REAL XI,XX,YY,ZZ,T(5)(5)
                        !Defines real variables and 5x5 matrix.
XX=-20;ZZ=10;XI=0.1     !Assigns values to the variables.
!---------------------- Assign values to T matrix --------------------
T(0)(0)=2;T(0)(1)=22;T(0)(2)=6;T(0)(3)=8;T(0)(4)=10; T(1)(0)=12;
T(1)(1)=-44;T(1)(2)=16;T(1)(3)=18; T(1)(4)=20;T(2)(0)=22;T(2)(1)=24;
T(2)(2)=26; T(2)(3)=68;T(2)(4)=30;T(3)(0)=12;T(3)(1)=34; T(3)(2)=59;
T(3)(3)=-38;T(3)(4)=40;T(4)(0)=92; T(4)(1)=44;T(4)(2)=46;T(4)(3)=10;
T(4)(4)=90
```

```
WHILE ZZ<70
XX=XX+XI                !Set conditions
ZZ=ZZ+XI
YY=MAP2B(XX,ZZ,T,-10,5,20,10
                        !MAP2B command where:
                        !XX and ZZ are variables.
                        !T is a 2x2 matrix that specifies points.
                        !-10 is the first XX point in the matrix.
                        !5 defines the fixed intervals between the XX points.
                        !20 is the first ZZ point in the matrix.
                        !10 defines the fixed intervals between the ZZ
                        !points.
DISP XX,ZZ,YY           !Displays values of XX, ZZ and YY.
END                     !Ends MAP2B.
STOP                    !Ends program
```

Table 57 shows the YY values as a function of XX and ZZ arguments.

Table 57 MAP2B Table

	XX							
ZZ		-30	-10	-5	0	5	10	50
	1	2	2	22	6	8	10	10
	20	2	2	22	6	8	10	10
	30	12	12	-44	16	18	20	20
	40	22	22	24	26	68	30	30
	50	12	12	34	59	-38	40	40
	60	92	92	44	46	10	90	90
	100	92	92	44	46	10	90	90

Key:

XX and ZZ values
Interpolated YY values outside the boundaries of array “Table”
“Table” array values

Figure 33 illustrates this **MAP2B** example on the Scope.

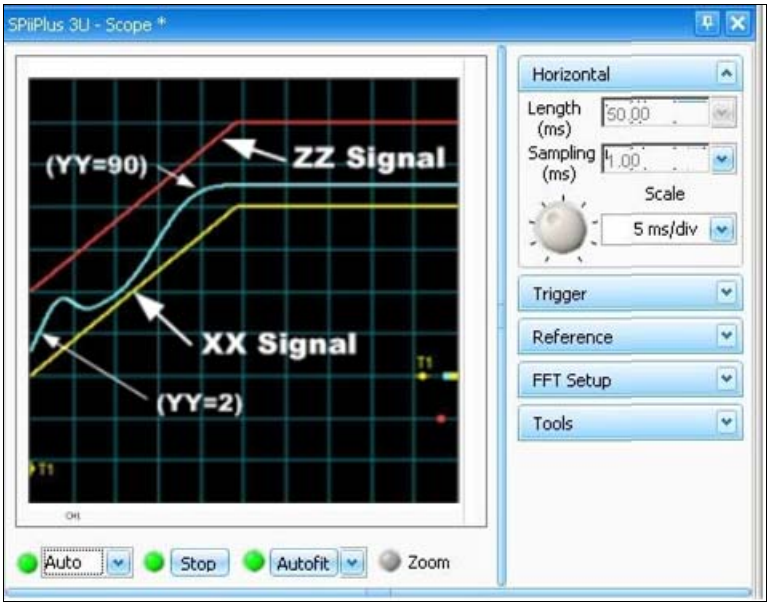


Figure 33 MAP2B Example on the Scope

4.6.16 MAP2N

Description

MAP2N returns a value from a two dimensional array of points according to the value of two input variables, with look-up arrays for each of the variables. The return values between the array-points are linearly interpolated. **MAP2N** is useful for dynamic error compensation.

Syntax

MAP2N(*XX,ZZ, table, X_table, Y_table*)

Arguments

XX	Real variable name or real expression.
ZZ	Real variable name or real expression.
table	The name of a real two-dimensional array that specifies points
X_table	A one-dimensional real array that specifies the values of **XX** that corresponds the point array.
Y_table	A one-dimensional real array that specifies the values of **ZZ** that corresponds the point array.

Return Value

MAP2N returns the linearly interpolated value from the array according to the value of the variables **XX** and **ZZ**.

Error Conditions

The function detects the following error conditions:

- Error 3072, Wrong array size, when **table** has only one dimension or **X_table** or **Y_table** have the wrong dimension. **X_table** must contain **M** elements and **Y_table** must contain **N** elements.
- The values in **X_table** or **Y_table** are not sequenced in ascending order.

Example

```
GLOBAL REAL XI,XX,YY,ZZ,XTABLE(5),YTABLE(5),TABLE(5)(5)
                        !Defines real variables, two arrays and a 5x5 matrix.
XX=-90;ZZ=0;XI=0.1      !Assigns values to the variables.
!---------------------- Assign values to XTABLE array --------------------
XTABLE(0)=13; XTABLE(1)=-8; XTABLE(2)=0; XTABLE(3)=12; XTABLE(4)=51
!---------------------- Assign values to YTABLE array --------------------
YTABLE(0)=16; YTABLE(1)=21; YTABLE(2)=80; YTABLE(3)=82; YTABLE(4)=113
!---------------------- Assign values to TABLE matrix --------------------
TABLE(0)(0)=2; TABLE(0)(1)=22; TABLE(0)(2)=6; TABLE(0)(3)=8; TABLE(0)(4)=10
TABLE(1)(0)=12; TABLE(1)(1)=-44; TABLE(1)(2)=16; TABLE(1)(3)=18;
TABLE(1)(4)=20; TABLE(2)(0)=22; TABLE(2)(1)=24; TABLE(2)(2)=26;
TABLE(2)(3)=68; TABLE(2)(4)=30; TABLE(3)(0)=12; TABLE(3)(1)=34;
TABLE(3)(2)=59; TABLE(3)(3)=-38; TABLE(3)(4)=40; TABLE(4)(0)=92;
TABLE(4)(1)=44; TABLE(4)(2)=46; TABLE(4)(3)=10; TABLE(4)(4)=90
WHILE ZZ<200
XX=XX+XI                !Set conditions
ZZ=ZZ+XI
YY=MAP2N(XX,ZZ,TABLE,XTABLE,YTABLE)
                        !MAP2N command where:
                        !XX and ZZ are variables.
                        !TABLE is a 5x5 matrix specifying mapping points.
                        !XTABLE is an array specifying XX points.
                        !YTABLE is an array specifying ZZ points.
DISP XX,ZZ,YY           !Displays values of XX, ZZ and YY.
END                     !Ends MAP2N.
STOP                    !Ends program
```

Table 58 shows the YY values as a function of XX and ZZ arguments.

Table 58 MAP2N Table

	XX							
ZZ		-90	-13	-8	0	12	51	100
	0	2	2	22	6	8	10	10
	16	2	2	22	6	8	10	10
	21	12	12	-44	16	18	20	20
	80	22	22	24	26	68	30	30
	82	12	12	34	59	-38	40	40
	113	92	92	44	46	10	90	90
	150	92	92	44	46	10	90	90

Key:

XX and ZZ values
Interpolated YY values outside the boundaries of array “Table”
“Table” array values

Figure 33 illustrates this **MAP2N** example on the Scope.

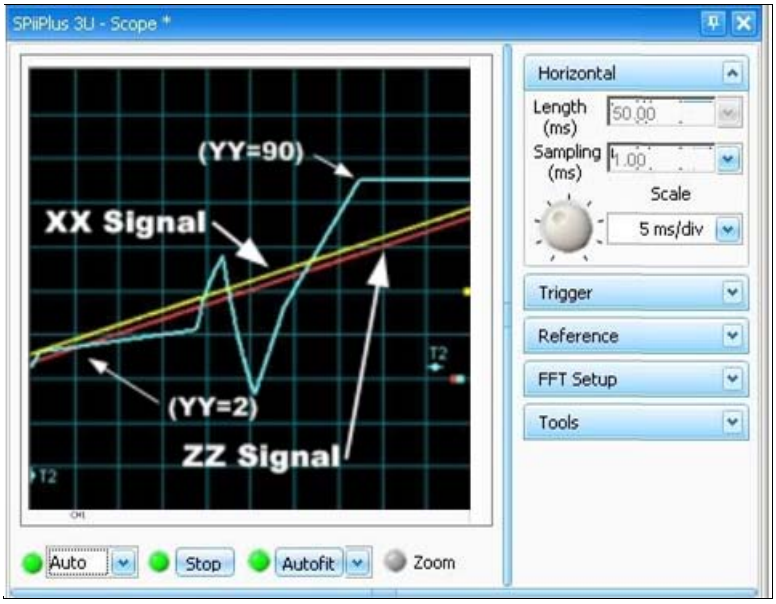


Figure 34 MAP2N Example on the Scope

4.6.17 MAP2NB

Description

MAP2NB returns a value from a two-dimensional array of points based on the input values of two variables, with a separate look-up array for one variable**,** and a separate look-up array for the other variable. The return values between the array-points are interpolated according to a third-order B-spline. **MAP2NB** is useful for dynamic error compensation.

Syntax

MAP2NB(*XX,ZZ, Table, X_Table, Y_Table*)

Arguments

XX	Real variable name or real expression.
ZZ	Real variable name or real expression.
table	The name of a real two-dimensional array that specifies points
X_table	A one-dimensional real array that specifies the values of **XX** that corresponds the point array.
Y_table	A one-dimensional real array that specifies the values of **ZZ** that corresponds the point array.

Return Value

MAP2NB returns the third-order B-spline interpolated value from the array according to the value of the variables **XX** and **ZZ**.

Error Conditions

The function detects the following error conditions:

- Error 3072, Wrong array size, when **table** has only one dimension or **X_table** or **Y_table** have the wrong dimension. **X_table** must contain **M** elements and **Y_table** must contain **N** elements.
- The values in **X_table** or **Y_table** are not sequenced in ascending order.

Example

```
GLOBAL REAL XI,XX,YY,ZZ,XTABLE(5),YTABLE(5),TABLE(5)(5)
                        !Defines real variables, two arrays and a 5x5 matrix.
XX=-90;ZZ=0;XI=0.1      !Assigns values to the variables.
!---------------------- Assign values to XTABLE array --------------------
XTABLE(0)=-13; XTABLE(1)=-8; XTABLE(2)=0; XTABLE(3)=12; XTABLE(4)=51
!---------------------- Assign values to YTABLE array --------------------
YTABLE(0)=16; YTABLE(1)=21; YTABLE(2)=80; YTABLE(3)=82; YTABLE(4)=113
!---------------------- Assign values to TABLE matrix --------------------
TABLE(0)(0)=2; TABLE(0)(1)=22; TABLE(0)(2)=6; TABLE(0)(3)=8; TABLE(0)(4)=10;
TABLE(1)(0)=12; TABLE(1)(1)=-44; TABLE(1)(2)=16; TABLE(1)(3)=18;
TABLE(1)(4)=20; TABLE(2)(0)=22; TABLE(2)(1)=24; TABLE(2)(2)=26;
```

```
TABLE(2)(3)=68; TABLE(2)(4)=30; TABLE(3)(0)=12; TABLE(3)(1)=34;
TABLE(3)(2)=59; TABLE(3)(3)=-38; TABLE(3)(4)=40; TABLE(4)(0)=92;
TABLE(4)(1)=44; TABLE(4)(2)=46; TABLE(4)(3)=10; TABLE(4)(4)=90
WHILE ZZ<200
XX=XX+XI                !Set conditions
ZZ=ZZ+XI
YY=MAP2NB(XX,ZZ,TABLE,XTABLE,YTABLE)
                        !MAP2NB command where:
                        !XX and ZZ are variables.
                        !TABLE is a 5x5 matrix specifying mapping points.
                        !XTABLE is an array specifying XX points.
                        !YTABLE is an array specifying ZZ points.
DISP XX,ZZ,YY           !Displays values of XX, ZZ and YY.
END                     !Ends MAP2N.
STOP                    !Ends program
```

Table 59 shows the YY values as a function of XX and ZZ arguments.

Table 59 MAP2NB Table

	XX							
ZZ		-90	-13	-8	0	12	51	100
	0	2	2	22	6	8	10	10
	16	2	2	22	6	8	10	10
	21	12	12	-44	16	18	20	20
	80	22	22	24	26	68	30	30
	82	12	12	34	59	-38	40	40
	113	92	92	44	46	10	90	90
	150	92	92	44	46	10	90	90

Key:

XX and ZZ values
Interpolated YY values outside the boundaries of array “Table”
“Table” array values

Figure 33 illustrates this **MAP2NB** example on the Scope.

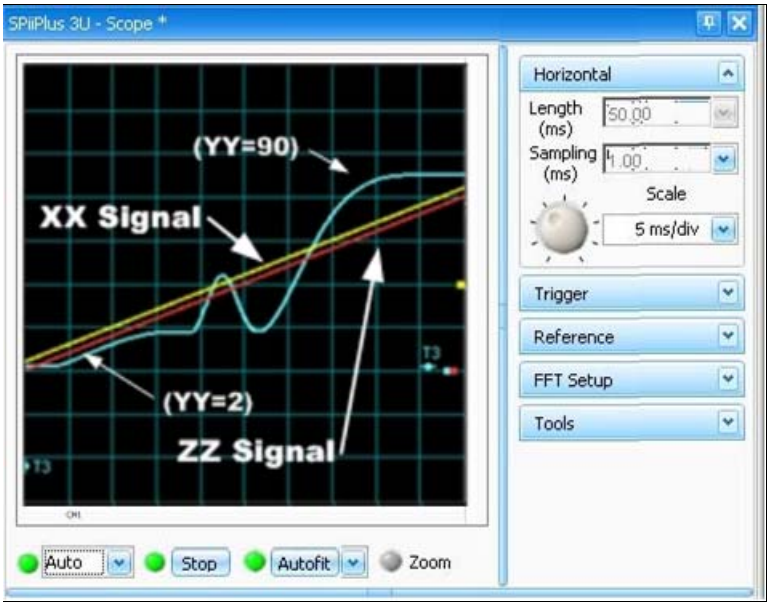


Figure 35 MAP2NB Example on the Scope

4.6.18 MAP2NS

Description

MAP2NS returns a value from a two-dimensional array of points based on the input values of two variables, with a separate look-up array for one variable**,** and a separate look-up array for the other variable. The return values between the array-points are interpolated according to a third-order Catmull-Rom spline. The **MAP2NS** function is useful for dynamic error compensation

Syntax

MAP2NS(*XX,ZZ, table, X_table, Y_table*)

Arguments

XX	Real variable name or real expression.
ZZ	Real variable name or real expression.
table	The name of a real two-dimensional array that specifies points
X_table	A one-dimensional real array that specifies the values of **XX** that corresponds the point array.
Y_table	A one-dimensional real array that specifies the values of **ZZ** that corresponds the point array.

Return Value

MAP2NS returns the third-order Catmull-Rom interpolated value from the array according to the value of the variables **XX** and **ZZ**.

Error Conditions

The function detects the following error conditions:

- Error 3072, Wrong array size, when **table** has only one dimension or **X_table** or **Y_table** have the wrong dimension. **X_table** must contain **M** elements and **Y_table** must contain **N** elements.
- The values in **X_table** or **Y_table** are not sequenced in ascending order.

Example

```
GLOBAL REAL XI,XX,YY,ZZ,XTABLE(5),YTABLE(5),TABLE(5)(5)
                        !Defines real variables, two arrays and a 5x5 matrix.
XX=-90;ZZ=0;XI=0.1      !Assigns values to the variables.
!---------------------- Assign values to XTABLE array --------------------
XTABLE(0)=13; XTABLE(1)=-8; XTABLE(2)=0; XTABLE(3)=12; XTABLE(4)=51
!---------------------- Assign values to YTABLE array --------------------
YTABLE(0)=16; YTABLE(1)=21; YTABLE(2)=80; YTABLE(3)=82; YTABLE(4)=113
!---------------------- Assign values to TABLE matrix --------------------
TABLE(0)(0)=2; TABLE(0)(1)=22; TABLE(0)(2)=6; TABLE(0)(3)=8; TABLE(0)(4)=10;
TABLE(1)(0)=12; TABLE(1)(1)=-44; TABLE(1)(2)=16; TABLE(1)(3)=18;
TABLE(1)(4)=20; TABLE(2)(0)=22; TABLE(2)(1)=24; TABLE(2)(2)=26;
TABLE(2)(3)=68; TABLE(2)(4)=30; TABLE(3)(0)=12; TABLE(3)(1)=34;
TABLE(3)(2)=59; TABLE(3)(3)=-38; TABLE(3)(4)=40; TABLE(4)(0)=92;
TABLE(4)(1)=44; TABLE(4)(2)=46; TABLE(4)(3)=10; TABLE(4)(4)=90
WHILE ZZ<200
XX=XX+XI                !Set conditions
ZZ=ZZ+XI
YY=MAP2NS(XX,ZZ,TABLE,XTABLE,YTABLE)
                        !MAP2NS command where:
                        !XX and ZZ are variables.
                        !TABLE is a 5x5 matrix specifying mapping points.
                        !XTABLE is an array specifying XX points.
                        !YTABLE is an array specifying ZZ points.
DISP XX,ZZ,YY           !Displays values of XX, ZZ and YY.
END                     !Ends MAP2N.
STOP                    !Ends program
```

Table 60 shows the YY values as a function of XX and ZZ arguments.

Table 60 MAP2NS Table

	XX							
ZZ		-90	-13	-8	0	12	51	100
	0	2	2	22	6	8	10	10
	16	2	2	22	6	8	10	10
	21	12	12	-44	16	18	20	20
	80	22	22	24	26	68	30	30
	82	12	12	34	59	-38	40	40
	113	92	92	44	46	10	90	90
	150	92	92	44	46	10	90	90

Key:	
	XX and ZZ values
	Interpolated YY values outside the boundaries of array “Table”
	“Table” array values

Figure 33 illustrates this **MAP2NS** example on the Scope.

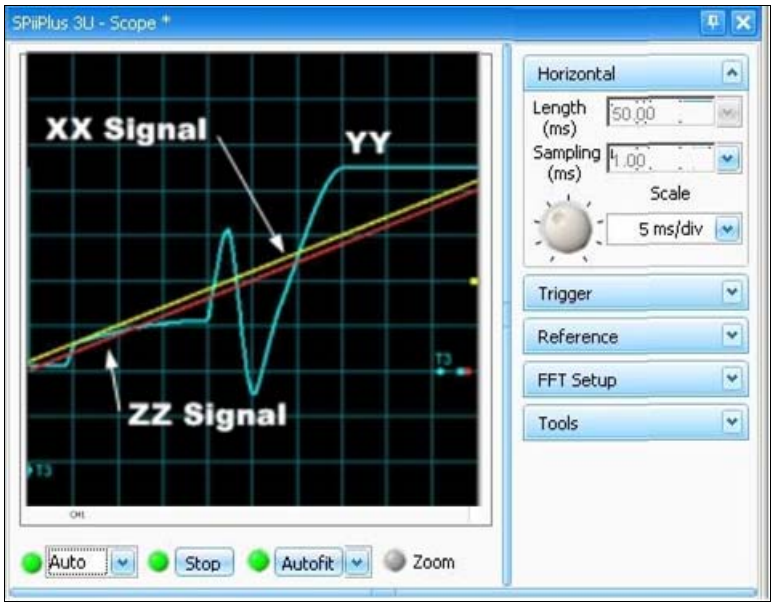


Figure 36 MAP2NS Example on the Scope

4.6.19 MAP2S

Description

MAP2S returns a value from a two-dimensional array of points based on the input values of two variables, the base values of the two variables, and the fixed defined intervals of the two variables. The return values between the array-points are interpolated according to a third order Catmull-Rom spline. The **MAP2S** function is useful for dynamic error compensation.

Syntax

MAP2S(*XX,ZZ, table, baseX, stepX, baseY, stepY*)

Arguments

XX	Real variable name or real expression.
ZZ	Real variable name or real expression.
table	The name of a real two-dimensional array that specifies points
baseX	A real number representing the value of **XX** that corresponds to the first point in the array.
stepX	A real number representing the value of **XX** that defines the fixed intervals between the array points.
baseY	A real number representing the value of **ZZ** that corresponds to the first point in the array.
stepY	A real number representing the value of **ZZ** that defines the fixed intervals between the array points.

Return Value

MAP2S returns the third order Catmull-Rom spline interpolated value from the array according to the value of variables **XX** and **ZZ**.

Error Conditions

The function detects the following error conditions:

- Error 3072, Wrong array size, when **table** has only one dimension.
- Error 3113, The step in the table is zero or negative, when the **stepX** or **stepY** arguments are zero or negative.

Example

```
GLOBAL REAL XI,XX,YY,ZZ,TABLE(5)(5)
                        !Defines real variables and 5x5 matrix.
XX=-20;ZZ=10;XI=0.1     !Assigns values to the variable.
!-------------------- Assign values to TABLE matrix ---------------------
TABLE(0)(0)=2; TABLE(0)(1)=22; TABLE(0)(2)=6; TABLE(0)(3)=8; TABLE(0)(4)=10;
TABLE(1)(0)=12; TABLE(1)(1)=-44; TABLE(1)(2)=16; TABLE(1)(3)=18;
TABLE(1)(4)=20; TABLE(2)(0)=22; TABLE(2)(1)=24; TABLE(2)(2)=26;
TABLE(2)(3)=68; TABLE(2)(4)=30; TABLE(3)(0)=12; TABLE(3)(1)=34;
```

```
TABLE(3)(2)=59; TABLE(3)(3)=-38; TABLE(3)(4)=40; TABLE(4)(0)=92;
TABLE(4)(1)=44; TABLE(4)(2)=46; TABLE(4)(3)=10; TABLE(4)(4)=90
WHILE ZZ<70
XX=XX+XI                !Set conditions
ZZ=ZZ+XI
YY=MAP2S(XX,ZZ,TABLE,-10,5,20,10)
                        !MAP2S where:
                        !XX and ZZ are variables.
                        !TABLE is a 2x2 matrix that specifies points.
                        !-10 is the first XX point in the matrix.
                        !5 defines the fixed intervals between the XX points.
                        !20 is the first ZZ point in the matrix.
                        !10 defines the fixed intervals between the ZZ
                        !points.
DISP XX,ZZ,YY           !Displays values of XX, ZZ and YY.
END                     !Ends MAP2S.
STOP                    !Ends program
```

Table 55 shows the YY values as a function of XX and ZZ arguments.

Table 61 MAP2S Table

	XX							
ZZ		-30	-10	-5	0	5	10	50
	1	2	2	22	6	8	10	10
	20	2	2	22	6	8	10	10
	30	12	12	-44	16	18	20	20
	40	22	22	24	26	68	30	30
	50	12	12	34	59	-38	40	40
	60	92	92	44	46	10	90	90
	100	92	92	44	46	10	90	90

Key:

XX and ZZ values
Interpolated YY values outside the boundaries of array “Table”
“Table” array values

Figure 33 illustrates this **MAP2S** example on the Scope.

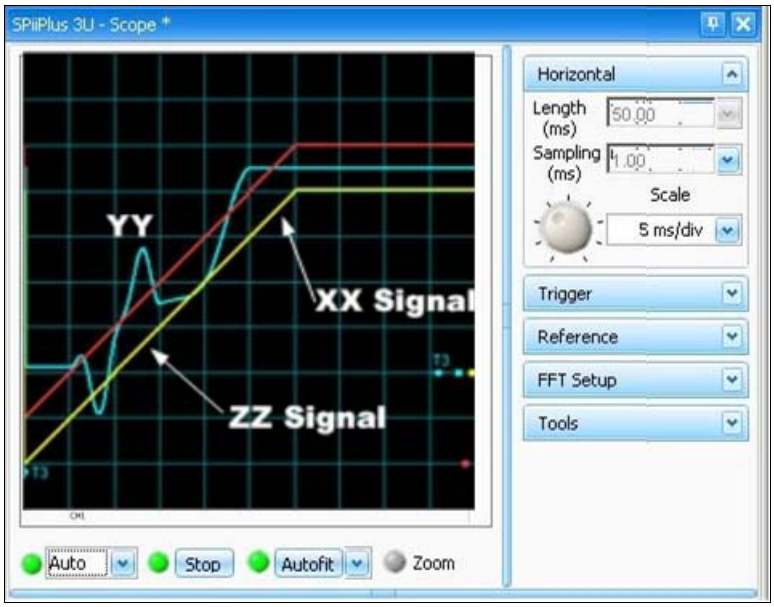


Figure 37 MAP2S Example on the Scope

4.6.20 MATCH

Description

MATCH calculates axis position (**APOS**) that matches the current reference position (**RPOS**), based on **CONNECT** between **APOS** and **RPOS** of the same axis. If there is no **CONNECT** command, the function returns the value of **RPOS**.

Syntax

real **MATCH** (*axis, from, to*)

Arguments

axis	The axis upon which to apply the MATCH function.
from	The start point to begin seaching for matching values of **APOS**.
to	The end point to stop seaching for matching values of **APOS**.

Comments

The function is useful only in the case of non-default connections.

The connection must be on the same axis.

The function succeeds if the unique root exists in the specified range. If there are several roots in the range, the function returns one of them. If the root does not exist, the function results an error.

Return Value

APOS that matches the current **RPOS** of the same axis.

Error Conditions

Error 3158 - The function cannot find a matching value. The **CONNECT** formula has no root or the root is not single.

Example

```
GLOBAL REAL ARRAY(6), XX
!-------------------- Assign values to ARRAY array --------------------
ARRAY(0)= 60; ARRAY(1)=40; ARRAY(2)=90; ARRAY(3)=-40; ARRAY(4)=60;
ARRAY(5)=10
VEL(0)=800              !Set VEL(0)
MFLAGS(0).17=1
CONNECT RPOS(0)=APOS(0)+MAP(APOS(0),ARRAY,100,200)
                        !The program executes a CONNECT between RPOS and APOS
                        !for axis X. Where RPOS(0) receives the value of
                        !APOS(0) added to the value of  a MAP function.
DEPENDS X,X
SET APOS(0)=0
SET RPOS(0)=MAP(APOS(0),ARRAY,100,200)
PTP X, 1300             !A PTP command sends the motor to 1300 counts,
                        !while the motor is in motion a MATCH function is
                        !calculated and displayed along with RPOS(0) and
                        !APOS(0).
WHILE MST(0).#MOVE
WAIT 300
XX=MATCH(X,0,1300)      !The MATCH function calculates the root of the
                        !current RPOS(0) based on the CONNECT formula.
DISP RPOS(0), APOS(0), XX
END
STOP                    !Ends program
```

The output of the program displays the following **RPOS(0)**, **APOS(0)** values, and the value of **MATCH**. Notice that **MATCH** values are very close to **APOS(0)** values:

RPOS(0)	APOS(0)	MATCH
273.76	228	226.4
554	472.8	471.2
684	717.6	716
1005.6	962.4	960.8
1215.6	1207.2	1205.6
1310	1300	1300

4.6.21 OUTP

Description

OUTP sends data values from an integer array to a specified channel. This function is useful to create an interface between the controller and special input devices such as a track-ball, mouse and various sensors. **OUTP** is also used when the controller acts as a master with the **MODBUS** protocol communication.

Before using **OUTP**, configure the relevant communication channel as a special communication channel using **SETCONF** function, key 302.

Each ASCII character is represented by its numerical value and is stored in a separate element of the array.

The user might have to define communication parameters for the special communication channel with **SETCONF** function keys 302, 303, 304, 309.

See also **INP**.

Syntax

int **OUTP**(*channel, variable, [start_index,] number, [timeout]*)

Arguments

channel	Communication channel index: ❑ 1 - serial communication channel COM1. ❑ 2 - serial communication channel COM2. ❑ 6 - Ethernet network (TCP) ❑ 7 - Ethernet network (TCP) ❑ 8 - Ethernet network (TCP) ❑ 9 - Ethernet network (TCP) ❑ 10 - Ethernet Point-to-Point (UDP) ❑ 12 - PCI bus ❑ 16 - communication channel with **MODBUS** slave ❑ 36 - Ethernet network (TCP) ❑ 37 - Ethernet network (TCP) ❑ 38 - Ethernet network (TCP) ❑ 39 - Ethernet network (TCP)
variable	User-defined integer array to which the data will be sent.
start_index	The index in the array from which to start. If **start_index** is omitted, the assignment starts from the first element of the array.The first received character is assigned to the array element with the specified index.
number	The number of characters to be transmitted to the variable array.
timeout	The function waits for output not more than the specified number of milliseconds. If **timeout** is omitted, the waiting time is not limited.

Return Value

The number of entities that have been transmitted.

Error Conditions

If the function fails, an error is generated.

4.6.22 RAND

Description
RAND generates a random number

Syntax
real **RAND**(*min, max[,seed]*)

Arguments

min	Lower boundary of the interval from which randomized number will be selected.
max	Upper boundary of the interval from which randomized number will be selected.
seed	Sets the random sequence generator. The number range is limited only by the controller register size - 32bit. If omitted, the controller uses ACSPL+ **TIME** variable as the seed.

Comments
RAND can generate only one random number per seed number (per interval). For generating a series of random numbers use **TIME** as the seed. In this case the series of random numbers will be in ascending order.

Return Value
The randomized generated number.

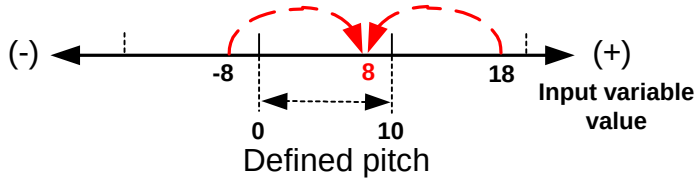
Error Conditions
None

Example
```
RAND (-45,45,TIME)
```

4.6.23 ROLL

Description

The **ROLL** function returns a result rolled-over to within a defined Pitch (range), as illustrated below.



Syntax

ROLL(*X, Pitch*)

Arguments

X	Variable name declared as real or a real expression.
Pitch	A range from 0 to **Pitch** (positive real value) where the return value will be rolled-over.

Return Value

X - if **X** falls within the range from 0 to Pitch.
or
|**X**|/**Pitch** - **FLOOR**(|**X**|/**Pitch**)*****Pitch**, if **X** is less than or greater than **Pitch**.

Error Conditions

None

Example:

```
YY=ROLL(XX,10)                    !Input variable XX is changing between -
                                  !20 to 20.
```

Figure 38 illustrates this **ROLL** example on the Scope.

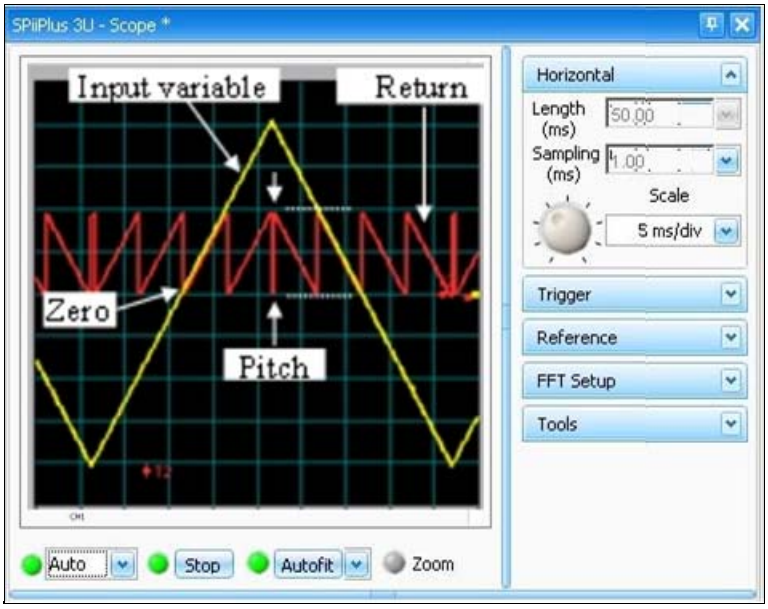


Figure 38 ROLL Example on the Scope

4.6.24 SAT

Description

SAT returns a result of defined saturation range

Syntax

real **SAT**(*X, Min, Max*)

Arguments

X	Variable name declared as real or a real expression.
Min	Low saturation value. Must be a real number.
Max	High saturation value. Must be a real number.

Return Value

X - if the **X** value falls inside the SAT range.
Min - if **X** value is less than the SAT range.
Max - if **X** value is greater than SAT range.

Error Conditions

None

Example

```
GLOBAL REAL YY,ZZ,XX
EYAL:
ZZ=-20;XX=0.1
WHILE ZZ<=20
ZZ=ZZ+XX
YY=SAT(ZZ,-5,15)
END
ZZ=20;XX=0.1
WHILE ZZ>=-20
ZZ=ZZ-XX
YY=SAT(ZZ,-5,15)
END
GOTO EYAL
STOP
```

Figure 39 illustrates this **SAT** example on the Scope.

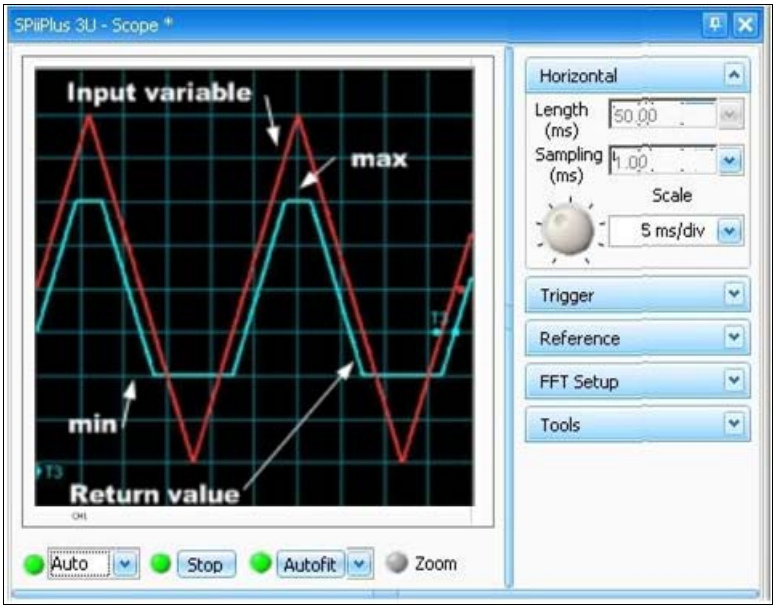


Figure 39 SAT Example on the Scope

5 Terminal Commands

Terminal commands are those commands that are specific to the SPiiPlus MMI Application Studio **Communication Terminal** utility and are not part of the ACSPL+, and cannot be incorporated into ASCPL+ programming. As soon as the command is received through one of the communication channels, it is executed.

This chapter covers all of the available Terminal commands.

5.1 Entering Terminal Commands

Terminal commands are entered in the **Communication Terminal** (the general structure of which is shown in **Figure 40**).

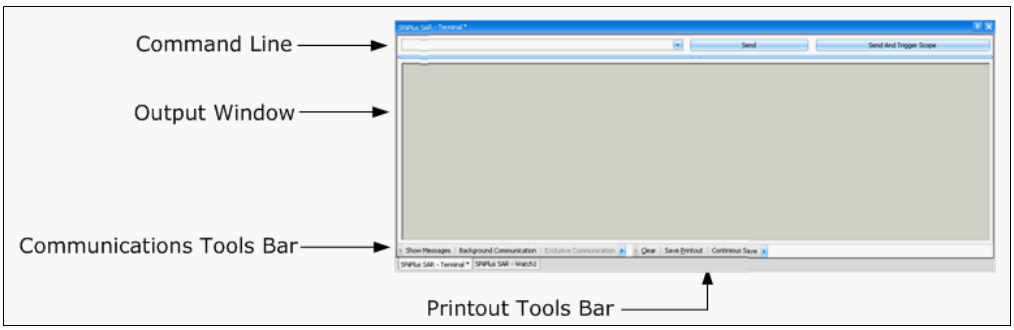


Figure 40 Communication Terminal Window

The **Communication Terminal** window is described in the *SPiiPlus MMI Application Studio User Guide*.

Note	
	Terminal commands are case sensitive.

Terminal commands are divided into:

- **Query Commands**
- **Program Management Commands**
- **System Commands**

5.2 Query Commands

Query commands are designated by the question mark (**?**) and are entered through the **Communication Terminal**.

Command & Syntax	Description	Example
?[*ACSPL+ variable*]	Returns the current value of a standard variable	**?TCPIP** – Returns the TCP/IP for the Ethernet port N1.
?[*ACSPL+ variable*], [*ACSPL+ variable*]...	Returns the current value of listed standard variables.	**?TCPIP**, **TCPPORT** – Returns the TCP/IP for the Ethernet port N1 and the TCP port number.
?[*buffer number*]: [*local user-defined variable*]	Returns the current values of a local user variable or array defined in a program buffer.	**?1:MY_VAR** – Returns the values of a local user-defined variable named MY_VAR.
?[*array_variable(index)*]	Retruns the current value of a specific element in the given array. The brackets enclosing the index are optional.	**?FPOS(0)** – Returns the feedback position of X axis. **?FPOS0:** Returns the feedback position of X axis.
?[*global user array_variable[(index)]*]	Returns the current values of a global user variable or array. Or the value of an element in the array if index is included.	**?ARRAY1** – Returns the values of a global user-defined array variable named ARRAY1. **?ARRAY1(3)** – Returns the value of the fourth element of a global user-defined array variable named ARRAY1.
?[*matrix_variable (row_index) (col_index)*]	Returns the current value(s) of an element in a two dimensional matrix. The indices may be entered as a range, e.g., (0,4) is the first through fifth, inclusively.	**?MATRIX** – Returns all of the values (in tablular format) of the two-dimensional array named MATRIX. **?MATRIX(1)(0)** – Returns the value of the second element in the first column of MATRIX. **?MATRIX(0)** – Returns all the values of the first row of MATRIX. **?MATRIX(0,2)(0,1)** – Returns a range of values, those of the first through third rows in the first and second columns of MATRIX.
?[*buffer number*]	Returns the current status and information about a program buffer	**?0** – Returns the current status and information about program buffer 0.
?#	Returns the current status of all program buffers.	
?$[*axis number*]	Returns the current status of the motor for the specified axis.	**?$16** – Returns the current status of the axis 16 motor.

Command & Syntax	Description	Example
?$	Returns the current status of all motors.	
?VR	Returns the firmware version	
?SN	Returns the controller serial number and hardware version - indicated by a letter.	
??[*error_code*]	Returns the error code number. If the error code is included in the query, returns error description.	**??3260:** Returns "Motor cannot start because the motor is disabled"
??[*variable_name*]	Returns a brief description of the variable.	**??CERRA** - Returns "Critical Position Error In Accelerating"

5.2.1 Default Query Formats

All the queries described above produce a variable report in a default format depending on the queried variable. In some cases the default format produces unsatisfactory results. For very large or small real values the output may appear misleading because very large or small values may require more positions than allocated by the default.

The default format for all real variables is similar to the C "%10G" format, where each real value is represented by 10 digits. The controller automatically chooses the position of decimal point and the number of digits right to the decimal point. If required, the controller uses an exponential format, like 3.14E-13.

The default format for all user integer variables and for all standard integer variables, except State flags, and I/O variables, is similar to the C "%10i" format that specifies 10 digits for each integer number.

State and Flag variables are reported in special format. Each State and Flag variable is a collection of bits. Each bit within the variable has its own function. The variable is reported in a format that displays the state of each bit with a short explanation.

The standard I/O arrays IN and OUT are reported in binary format.

5.2.2 Predefined Query Output Formats

The controller provides several predefined formats that can be used instead of the default format for querying variables:

Format	Description
?D/	Decimal format. This format is identical to the default format for integer variables. When applied to a state variable, the format displays the decimal presentation of the variable. C-equivalent: %10i.
?X/	Hexadecimal format. When applied to an integer variable, this format displays the hexadecimal presentation of the variable. C-equivalent: %08X
?B/	Binary format. This format is identical to the default format for the **IN** and **OUT** variables. When applied to an integer variable, the format displays the binary presentation of the variable.
?E/	Extended format. This format is useful for very large or small real values, when the default format produces ambiguous results because the default does not provide enough positions to display very large or very small numbers. When applied to a real variable, the format displays each value in 20 positions. C-equivalent: %20G

Examples

Display the motor state in decimal format:

?D/ MST

3 15 15 3 0 0 0 0

?X/Y_MST

0 0 0 0 0 0 0 0

Display the state of motor x in binary format

?B/ MST

00000000,00000000,00000000,00000011

Display the status of variable UserReal in extended format

?E/UserReal

1.00000000000001

5.2.3 User-Defined Query Output Format

If the default format or any other predefined format is not suitable for the user needs, the user may specify a format using C notation. The specification is placed just before the variable name in curled brackets. The specification applies only to the value of the name that is specified in the command. If an array is queried, the specification applies to each element of the array.

C notation provides an unlimited number of possible formats.

Examples

Format	Description
?{%12.3f}FPOS	The motor feedback values for all axes are displayed in 12 digits, fixed decimal point, 3 digits after the point. The same format applies to all 8 values.
?{%8.0f}X_FPOS	8 digits, no decimal point, no fraction digits.
?{XFPOS = %8.0f}X_FPOS	The response will look like **XFPOS = 1234.**
?{%08X} X_MST	8 digits, hexadecimal format with leading zeros

5.3 Program Management Commands

Program Management commands are used for:

- Controlling program execution
- Viewing and editing program content

The Program Management commands are designated by the pound (#) character.

5.3.1 Program Management Command Arguments

Program Management commands can take two types of arguments:

Buffer Designation

Buffer designation can be specified in two forms:

- An integer number, between 0 and 16, that addresses a specific buffer (16 addresses the D-buffer)
- The pound character, **#,** that addresses all program buffers in one command

  Only the following commands can address all buffers:
  - L – List
  - C – Compile
  - S, SR – Stop, Stop and Reset
  - P – Pause
  - F, FI – Find
  - BR – Reset Breakpoints

  All other commands operate with one buffer only, and must specify the buffer number.

Line Designation

A line designation can appear in one of three forms, as shown in the following table:

Table 62 Line Designation

Line Qualifier Type	Description
Single number	Specifies only one line.
Two numbers seperated by comma	Specifies a range of lines. If the second number is larger than a total number of lines in the program, the list range spans the last program line.
Label preceded by a slash (/) character	Specifies the line by a designated label (the text following the slash).
Label preceded by a slash (/) character, then a comma and number	Specifies a range of lines starting from the line with a designated label.

Examples

The following are examples of using the **L** command.

List line 4 in buffer 3.

```
#3L4
4: till ^MST(0).#MOVE
```

List lines from 1 to 3 inclusively in buffer 5.

```
#5L1,3
1: movePTP:
2: VEL(0) = 20000
3: ptp  0, 4000
```

List the line that contains the label **MovePTP** in buffer 5.

```
#5L/MovePTP
1: movePTP:
```

List 3 lines, starting from the label **movePTP** in buffer 5.

```
#5L/MovePTP, 3
1: movePTP:
2: VEL(0) = 20000
3: ptp  0, 4000
```

5.3.2 Program Buffer Commands

5.3.2.1 Open/Close Buffer (#)

Description

The # (Open/Close Buffer) command is used to open and close a buffer for the purpose of entering code.

Syntax

#buffer_number[I][line_number]

Arguments

buffer_number	**buffer_number** qualifier in the command specifies the buffer, a number between 0 and 16.
I	Optional, if included, opens the buffer for insertion of code.
line_number	Optional, if included, the command opens the buffer specified by **buffer_number** and sets the insert line before the line specified by **line_number**. **line_number** can be specified in any of the three forms listed in **Table 62.**

Comments

The **#** command opens the buffer and sets the insert line as follows:

- If the buffer is empty, the insert line is 1.
- If the buffer already contains a program, the insert line is set after the last line of the program.

Only one buffer can be opened at a time.

To close the buffer, # is entered without the buffer number (since only one buffer is open, the controller knows which buffer to close.

Examples

The following are examples of opening and closing buffers.

`#0`	Open buffer 0
`0:00001>`	Buffer 0 is empty
`#3`	Open buffer 3
`3:00006>`	This indicates that buffer 3 contains 5 lines. The insert line is set to 6.
`#3I`	Open buffer 3 for insertion
`3:00006>`	The command does not specifies a line qualifier. It is identical to the command #3.
`#3I2`	Open buffer 3 and set insertion point prior to line 2.
`3:00002>`	Insert line is set before line 2

`#`	Close the buffer
`:`	The prompt indicates that all buffers are closed

When a program buffer is open, ACSPL+ commands are stored in the buffer (and are not executed immediately). The controller checks the syntax of inserted ACSPL+ lines and immediately reports any errors detected.

The following is an example of an editing session:

`#0`	Open buffer 0
`0:00001>`	Buffer 0 is empty
`VEL(0) = 20000`	Enter the program lines sequentially. After each line, the insert line number is incremented
`0:00002>`	
`ptp  X, 4000`	
`0:00003>`	
`till ^MST(0).#MOVE`	
`0:00004>`	
`stop`	
`0:00005>`	
`#0L`	List the program
`1: VEL(0) = 20000`	
`2: ptp  0, 4000`	
`3: till ^MST(0).#MOVE`	
`4: stop`	
`0:0005>`	The buffer remains open
`#0I1`	Change the insert line number to 1
`0:0001>`	Insert line is set before the first line
`MovePTP:`	Insert label
`0:0002>`	
`#0L`	List the program
`1: MovePTP:`	
`2: VEL(0) = 20000`	
`3: ptp  0, 4000`	
`4: till ^MST(0).#MOVE`	
`5: stop`	
`0:0002>`	The buffer remains open
`#`	Close the buffer

5.3.2.2 D

Description

The **D** (Delete) command deletes the specified lines in the buffer.

Syntax

#buffer_numberDline_number[,line_number]

Arguments

buffer_number	**buffer_number** qualifier in the command specifies the buffer, a number between 0 and 16.
line_number	**line_number** in the **D** command is obligatory. **line_number** can be: ❑ A single number, specifying one specific line, or ❑ Two numbers separated by comma that specify a range of lines. If the second number is larger than a total number of lines in the program, the delete range includes the last program line.

Comments

If a buffer is open, the **D** command that addresses the buffer shifts the insert line to before the first undeleted line.

Example

`#0L`	Open buffer 0
`0:00001>`	Buffer 0 is empty
`#3`	Open buffer 3
`3:00006>`	Buffer 3 contains 5 lines. The insert line is set to 6.
`#3I`	
`3:00006>`	The command does not specifies a line qualifier. It is identical to the command #3
`#3I2`	
`3:00002>`	Insert line is set before line 2
#0D3	Delete line 3 in buffer 0
`#`	Close the buffer
`:`	The prompt indicates that all buffers are closed

5.3.2.3 F/FI

Description

The **F**/**FI** (Find/Find Case-sensitve) commands are used to search for a specific text in a specified buffer or in all buffers.

Syntax

#buffer_number{F|FI}/search_string [,line_number]

Arguments

buffer_number	**buffer_number** qualifier in the command specifies the buffer, a number between 0 and 16.
search_string	The text being sought.
line_number	Optional, if included, **line_number** defines the start line for the search. Otherwise, the search starts from the first line.

Comments

search_string must be specified as a label, that is, it must be preceded by a slash (/), or as a label and number separated by comma. **search_string** can be any text, such as, a variable name, ACSPL+ command, constant, label or keyword.

The search terminates when the first entry of the specified text is found, or the buffer end is reached. The command reports the line that contains the text, or an error message if the text was not found.

To find the next entry, the user must execute the command again, specifying the new start line number of the reported line plus one.

If the # character is specified instead of the buffer number, the search command addresses all buffers. In this case the command finds the first entry in each buffer.

Examples

The following are examples of using the **F** command.

`#0F/X`	Find “X” in buffer 0
`0002`	Response: found entry in line 2
`#0F/X,3`	Find the next entry, starting from line 3
`0003`	Response: found entry in line 3
`#0F/X,5`	Find the next entry, starting from line 5
`?1078`	Response: No more entries

`##F/stop`	Find **stop** in all buffers
`Buffer #0: no matches`	
`Buffer #1:  4: stop`	Entry is found in line 4 of Buffer 1
`Buffer #2: no matches`	
`Buffer #3:  4: stop`	Entry is found in line 4 of Buffer 3
`Buffer #4:  1: stop`	Entry is found in line 1 of Buffer 4
`Buffer #5: no matches`	
`Buffer #6: no matches`	
`Buffer #7: no matches`	
`Buffer #8:  238: stop`	Entry is found in line 238 of Buffer 8
`Buffer #9: no matches`	
`Buffer #10: no matches`	

5.3.2.4 L

Description

The **L** (List) command is used for displaying a program listing.

Syntax

#buffer_numberL[line_number]

Arguments

buffer_number	**buffer_number** specifies the buffer, a number between 0 and 16; or you can use the pound (#) to designate all buffers.
line_number	Optional, if included, **line_number** defines a specific line to be listed. Otherwise, the search starts from the first line. **line_number** can be specified in any of the three forms listed in **Table 62.**

Comments

The listing contains all program lines preceded by line numbers. Each line appears exactly as it was inserted. No automatic formatting is provided.

To address all buffers the # character is used instead of the buffer number, for example, the command **##L** provides a listing of all programs in all buffers. If **line_number** is included and a buffer does not contain the specified line number, only the buffer number is listed.

If the buffer is empty, the list includes the buffer designation followed by the first line (0) which is blank.

Note	
	It is recommended that you place a remark with a short program description in the first line of each program. This enables using the command ***##L1*** *to get quick information about all loaded programs*

Examples

The following are examples of the **L** command.

Example 1:

Provide a program listing for buffer #3:

```
#3L
1: MovePTP:
2: VEL(0) = 20000
3: ptp  X, 4000
4: till ^MST(0).#MOVE
5: stop
```

Example 2:

Provide a program listing for the first line in all buffers:

```
##L1                                  List the contents of line 1 in all buffers.
Buffer 0                             Response
       0:
Buffer 1
       1: ! Homing of all axes
Buffer 2
       1: ! Registration motion
Buffer 3
       1: MovePTP:
Buffer 4
       1: ! PLC program
Buffer 5
       0:
Buffer 6
       0:
Buffer 7
       0:                            Buffer 5 is empty
Buffer 8
       0:
Buffer 9
       0:                            Buffer 6 is empty

                                     Buffer 7 is empty

                                     Buffer 8 is empty

                                     Buffer 9 is empty
```

5.3.2.5 PROTECT

Description
The **PROTECT** command is used to activate standard configuration variables protection, that is, once issued, no changes can be made to the values of the configuration variables.

Syntax
#PROTECT [“password”]

Comments
PROTECT can be specified with a password. A password can be any sequence of alpha-numeric characters enclosed in quotes (“”). The sequence can contain any printable or non-printable character, except the quotes.

If the controller was protected with a **password**, it accepts the **UNPROTECT** command only with the same password. Therefore, the application cannot be opened without knowing password.

PROTECT can also be specified without a password. In this case **UNPROTECT** does not require password either.

5.3.2.6 UNPROTECT

Description
The **UNPROTECT** command disables application protection (returns the controller to configuration mode).

Syntax
#UNPROTECT [“password”]

Comments
If **#PROTECT** was applied with a password, the same password must accompany the corresponding **#UNPROTECT** command. If a password was not included in the **PROTECT** command, the **#UNPROTECT** command does not need a password.

If the controller is in protected mode, **RESET** can be applied to delete a password-protected application if the password is unknown. In this case, the application cannot be viewed or saved, but the controller will be reset to the factory-defaults.

Example

```
#PROTECT “123MyPassword”
```

5.3.3 RESET

Description
The **RESET** command is used to reset the controller to factory default state.

Syntax
#RESET

Comments
The **RESET** command can be issued even if the application is in the Protected mode (see **PROTECT**), in which case the password, if included, is not needed.

5.3.4 Listing Program Variables

There are three types of variables:

- ACSPL+ Variables – variables contained in the ACSPL+ language set
- SP Variables – variables incorporated in the controller
- User-Defined Variables – variables that have been declared by the user

For each type of variable there is a Communication Terminal command for listing them.

5.3.4.1 VGR

Description
The **VGR** command lists the categories within which the ACSPL+ variables are grouped. The categories of the ASCPL+ variables are:

- Axis_State
- Monitoring
- Motion
- Safety_Control
- Inputs_Outputs
- Program_Execution_Control
- System_Configuration
- Axis_Configuration
- Communication
- Commutation
- Data_Collection
- Servo_Loop
- Miscellaneous
- Obsolete

Syntax
#VGR [group_name]

Arguments

group_name	One of the ACSPL+ variable categories.

Comments
If **group_name** is omitted, the command lists only the categories. If **group_name** is included, the command lists the ACSPL+ variables within the category.

Note	
	The category must be entered in exactly the same format as given in the list above.

5.3.4.2 VSD

Description
The **VSD** command lists all ACSPL+ variables with a short description.

Syntax
#VSD [group_name]

Arguments

group_name	One of the ACSPL+ variable categories (see **VGR**).

Comments
When **group_name** is included in the VSD command, the ACSPL+ variables within the specified category and a brief description of each variable is listed.

5.3.4.3 VS/VSG

Description
The **VS**/**VSG** commands are used to list the variables that are incorporated in the ACSPL+ language set.

Syntax
#VS

#VSG [group_name]

Arguments

group_name	One of the ACSPL+ variable categories (see **VGR**).

Comments
When **group_name** is included in the **VSG** command, the names of the ACSPL+ variables within the specified category are listed.

Example

The following is an example of the **VS** command.

#VS	List ACSPL+ variable names
	Response

```
ACC
AERR
AFLAGS
AIN
AOUT
APOS
AST
BAUD
BOFFTIME
BONTIME
MASK
FPOS
FVEL
FVFIL
GACC
GETIME
GJERK
GMOT
GMQU
GMTYPE
```

Note	
	For brevity, only a portion of the response is shown here.

5.3.4.4 VSF/VSGF

Description
Both the **VSF** and **VSGF** commands, in addition to the global variable names, display the variable type, the number of elements (for arrays only), address of the variable in the controller memory and the step between array elements (for arrays only).

Syntax
#VSF

#VSGF [group_name]

Arguments

group_name	One of the ACSPL+ variable categories (see **VGR**).

Comments
When **group_name** is included in the **VSGF** command, the names of the ACSPL+ standard variables within the specified category and their details are listed.

5.3.4.5 VG/VGF

Description
The **VG** command lists all global variable names in the system.

The **VGF** command, in addition to the global variable names, lists the variable type, the number of elements (for arrays only), address of the variable in the controller memory and the step between array elements (for arrays only).

Syntax
#[buffer_no]VG

#[buffer_no]VGF [variable_name]

Arguments

buffer_no	A number ranging from 0 to 16, representing a specific buffer.
variable_name	A specific ACSPL+ variable.

Comments
If **buffer_no** is included, **VG** and **VGF** list all the global variables in the specified buffer.

If **variable_name** specifying an ACSPL+ variable is included, **VGF** lists the details just for the specified variable

5.3.4.6 VL/VLF

Description
The **VL** command lists all local variable names in the system.

The **VLF** command, in addition to the local variable names, lists the variable type, the number of elements (for arrays only), address of the variable in the controller memory and the step between array elements (for arrays only).

Syntax
#[buffer_no]VL

#[buffer_no]VLF [variable_name]

Arguments

buffer_no	A number ranging from 0 to 16, representing a specific buffer.
variable_name	A specific ACSPL+ variable.

Comments
If **buffer_no** is included, **VL** and **VLF** list all the local variables in the specified buffer.

If **variable_name** specifying an ACSPL+ variable is included, **VGF** lists the details just for the specified variable.

5.3.4.7 V/VF

Description
The **V**/**VF** (List User-Defined Variable Names only/List User-Defined Variables with Description) commands are used to list the user-defined variables that are found in compiled programs.

Syntax
#[buffer_number]V

#[buffer_number]VF

Arguments

buffer_no	A number ranging from 0 to 16, representing a specific buffer.

Comments
If **buffer_no** is not specified, the list includes the user-defined variables in all compiled buffers. The **V** command only displays the names of the user-defined variables. The **VF**, on the other hand, in addition to the variable names, it displays the variable type, the number of elements (for arrays only), address of the variable in the controller memory and the step between array elements (for arrays only).

The list can be saved to a file by clicking **Save** in the **Communication Terminal** window.

Examples

The following are examples of the **V** and **VF** commands.

```
#9V                                  Provide a list of user variables in buffer 9
                                     Response
ITIME  Global
TS_AMP1Local
TS_AMP0Global
#9VF                                 Provide a list of user variables in buffer 9 with
                                     additional information
                                     Response
ITIME  Globalreal@00DA0C50
TS_AMP1Localint@00DA1A30
TS_AMP0 Globalint@00DA0C80
```

5.3.4.8 VSP

Description

The **VSP** (List Servo Processor Variables) command provides a list of the SP variables that are defined in the program in the specified SP.

Syntax

#VSPservo_number

Arguments

servo_number	**servo_number** is number of the Servo Processor.

Comments

Each variable name in the list is accompanied by an SP address of the variable.

The list can be saved to a file by clicking **Save** in the Terminal window.

Example:

```
#VSP0                                List the variables in SP 0
                                     Response
_RMS_SUM                    @088
A_RMS_SUM                    @089
X_NOTCH_OUT_PR_H             @08A
X_NOTCH_OUT_PR2_H            @08B
X_NOTCH_IN_PR                @08C
X_NOTCH_IN_PR2               @08D
X_NOTCH_FRC                  @08E
X_NOTCH_FRC_L                @08F
A_NOTCH_OUT_PR_H @091
A_NOTCH_IN_PR                @092
```

5.3.4.9 VST/VSGT

Description
Both the **VST** and **VSGT** commands display a list of ACSPL+ variables to which **PROTECT** can be applied.

Syntax
#VST

#VSGT [group_name]

Arguments

group_name	One of the ACSPL+ variable categories (see **VGR**).

Comments
When **group_name** is included with the **VSGT** command, the ACSPL+ variables within the specified category are listed.

5.3.4.10 VSTF/VSGTF/VSDT

Description
The **VSTF**, **VSGTF**, and **VSDT** commands all list the variable names, the variable type, the number of elements (for arrays only), address of the variable in the controller memory and the step between array elements (for arrays only) of those ACSPL+ variables to which **PROTECT** can be applied

Syntax
#VSTF

#VSGTF [group_name]

#VSDT [group_name]

Arguments

group_name	One of the ACSPL+ variable categories (see **VGR**).

Comments
When **group_name** is included with the **VSGTF** or **VSDT** command, the ACSPL+ variables within the specified category are listed.

5.3.4.11 VGV

Description
The **VGV** command is used to remove global variables that have been set via **Communication Terminal**.

Syntax
#VGV [global_var]

Arguments

global_var	Name of a global variable

Comments
If **global_var** is included, **VGV** removes only this variable; otherwise it removes all of them.

5.3.5 Program Handling Commands

These commands are used for compiling, executing, pausing and halting the program.

The commands in this set are:

- **C** - Compile Program
- **X** - Execute Program
- **S/SR** - Stop/Stop & Reset Program
- **P** - Pause Program

The following diagram shows the program buffer states and the Communication Terminal commands that affect the buffer states:

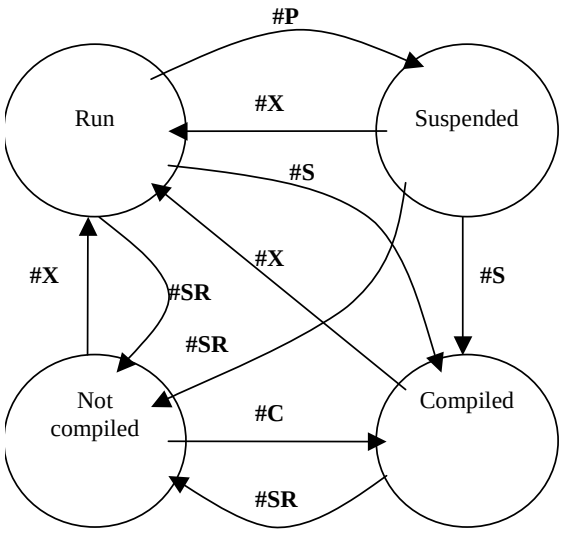


Figure 41 Interaction of Program Buffer States

The program buffer enters the Not Compiled state:

- After any change in the program text.
- Following execution of the #**S** (Stop) or #**R** (Reset) command.

The program buffer enters the **Compiled** state:

- Following execution of the #**C** (Compile) command the program buffer is transferred from the **Not Compiled** to the **Compiled** state.
- The #**S** (Stop) command transfers the program buffer from the Run or Suspended state to Compiled state.
- Following program termination with an ACSPL+ **STOP** command (or **RET** if an autoroutine was executed).
- When the program fails due to an error.

The program buffer enters the **Run** state:

- When the #**X** (Execute) command is issued.
- When another program executes a **STOP** command, or autoroutine condition is satisfied.

5.3.5.1 C

Description

The **C** (Compile) command compiles a program in the buffer or all programs in all buffers, depending on the buffer qualifier.

The **C** command must not include a line qualifier and is prohibited when the buffer is in the Run or Suspended states.

Syntax

#buffer_numberC

Arguments

buffer_number	**buffer_number** specifies the buffer, a number between 0 and 16; or you can can use the pound (#) to designate all buffers.

Comments

The **C** command is not obligatory in order to execute a program. When the **X** (Execute) command is issued, the controller automatically compiles the program if it was not previously compiled. However, a separate compilation step is required in the following cases:

- To check the program correctness without executing it.
- The program is not intended for direct starting, but contains autoroutines. The autoroutines are ready for execution only after compilation.
- The program is intended for starting from another program by the **START** command. The program started by the **START** command must be compiled before the **START** command can be executed.

If the program is successfully compiled, the controller prints a short report of how many lines were compiled. If an error was encountered, the controller reports the error code and the line number in which the error was found.

Examples
The following are examples of the **C** command.

#0C	Compile the program in buffer 0
5 lines compiled	Response, the program was compiled successfully
#9C	Compile the program in buffer 9
?2026 in line 16	Response, Error 2026 was found in line 16
??2026	Explain error 2026.
Undefined variable name	Response
##C	Compile the programs in all buffers
	Response:
Buffer 0: 5 lines compiled	The program was compiled successfully
Buffer 1: 18 lines compiled	The program was compiled successfully
Buffer 2: empty	The buffer is empty
Buffer 3: 5 lines compiled	The program was compiled successfully
Buffer 4: empty	The buffer is empty
Buffer 5: empty	The buffer is empty
Buffer 6: empty	The buffer is empty
Buffer 7: empty	The buffer is empty
Buffer 8: empty	The buffer is empty
Buffer 9: ?2026 in line 16	Error 2026 was found in line 16

5.3.5.2 X

Description
The **X** (Execute) command starts a program in a specific buffer, and can be executed in any program state except the **Run** state.

Syntax
#buffer_numberX[line_number]

Arguments

buffer_number	**buffer_number** qualifier in the command specifies the buffer, a number between 0 and 16.
line_number	Optional, **line_number**can be a line number or a label. Execution starts from the specified line. If **line_number** is omitted, the program starts from the first line.

Comments

buffer_number must specify one buffer only.

If the state of the program is **Not Compiled**, the controller first compiles the program and then starts it. If an error is encountered during compilation, the program does not start.

If the state of the program is **Suspended**, the X command resumes the program execution. In this case the command must not contain **line_number** because upon execution the program resumes from the point where the it was suspended.

Example

`#1X`	Execute the program in buffer 1
`?1`	Query status of buffer 1
`Buffer 1: 18 lines, running at line 7`	Response

5.3.5.3 S/SR

Description

The **S/SR** Commands are used for terminating program execution:

- **S** - Stop

  The **S** command terminates program execution in a buffer or the execution of all programs in all buffers.

- **SR** - Stop and Reset

  The **SR** (Stop and Reset) command terminates program execution in a buffer or the execution of all programs in all buffers, and resets the buffer or all buffers to the **Not Compiled** state. The command provides the de-compile function, which is useful if the program contains autoroutines that are ready to start when the buffer is in the **Compiled** state.

Syntax

#buffer_number{S|SR}

Arguments

buffer_number	**buffer_number** specifies the buffer, a number between 0 and 16; or you can use the pound (**#**) to designate all buffers.

Comments

If **buffer_number** is omitted**,** this will stop, or stop and reset all programs in all buffers, or you can use the # character as the **buffer_number**, for example, ##**S**, which will do the same.

Program termination commands must not include **line_numbers**.

The **S** command can be issued in any program state.

Note	
	The issuance of the **SR** *command effectively prevents the activation of autoroutines.*

Example

`#1S`	Terminate the program in buffer 1
`?1`	Query status of buffer 1
`Buffer 1: 18 lines, terminated in line 7`	Response
`#1SR`	Reset the program in buffer 1
`?1`	Query status of buffer 1
`Buffer 1: 18 lines, not compiled`	Response
`##SR`	Reset all programs in all buffers

5.3.5.4 P

Description

The **P** (Pause) command suspends program execution in a buffer.

Syntax

#buffer_numberP

Arguments

buffer_number	**buffer_number** specifies the buffer, a number between 0 and 16; or you can use the pound character, #, to designate all buffers.

Comments

Generally **buffer_number** refers to one buffer only. The # character may be used instead of a buffer number, for example, **##P**, in which case the execution of all programs in all buffers is suspended.

Pause commands must not include **line_numbers**.

Pause commands are allowed in any program state, but in all states other than **Run** the command has no effect.

If the program is in the **Suspended** state, the **X** command resumes execution. The **S - Stop** command transfers the buffer to the **Compiled** state. The **SR - Stop and Reset** command transfers the buffer to the **Not Compiled** state.

Example

`#1P`	Suspend the program in buffer 1
`?1`	Query status of buffer 1
`Buffer 1: 18 lines, suspended in line 7`	Response
`#1SR`	Reset the program in buffer 1
`?1`	Query status of buffer 1
`Buffer 1: 18 lines, not compiled`	Response
`##P`	Suspend all programs in all buffers

5.3.6 Debug Commands

The following debug commands are supported:

- **XS** - Execute one program line
- **XD** - Execute program in debug mode
- **BS** - Set breakpoint at specified line
- **BR** - Reset breakpoint

5.3.6.1 XS

Description

The **XS** (Execute one step) command executes one program line.

Syntax

#buffer_numberXSline_number

Arguments

buffer_number	**buffer_number** specifies the buffer, a number between 0 and 16.
line_number	**line_number** gives the line to be executed, it can be a line number or a label.

Comments

The **buffer_number** qualifier in the command must specify one buffer only.

After executing the specified **line_number**, the buffer automatically enters the **Suspended** state.

5.3.6.2 XD

Description
The**XD** (Execute in Debug mode) command executes the program up to the next breakpoint (see **BS**).

Syntax
#buffer_numberXD

Arguments

buffer_number	**buffer_number** specifies the buffer, a number between 0 and 16.

Comments
The **buffer_number** qualifier in the command must specify one buffer only.

The command is similar to the **X** command. The difference is that the **X** command ignores breakpoints in the program. If the program is started by the **XD** command, it will stop when it reaches a breakpoint. At the breakpoint the program transfers to the **Suspended** state and can be started again by the **X**, **XS**, or **XD** commands.

5.3.6.3 BS

Description
The **BS** (Set Breakpoint) command sets a breakpoint at the specified line.

Syntax
#buffer_numberBSline_number

Arguments

buffer_number	**buffer_number** specifies the buffer, a number between 0 and 16.
line_number	**line_number** specifies the line at which to set the breakpoint, it can be a line number or a label.

Comments
The **buffer_number** qualifier in the command must specify one buffer only.

Any number of breakpoints can be set in a program. For breakpoints to be active, the program must be started with the **XD** command.

Note	*In a program listing, the lines with breakpoints are indicated by an asterisk.*

5.3.6.4 BR

Description

The **BR** (Reset Breakpoint) command resets the breakpoint at the specified line or all breakpoints.

Syntax

#buffer_numberBR[line_number]

Arguments

buffer_number	**buffer_number** specifies the buffer, a number between 0 and 16; or you can use the pound (#) to designate all buffers.
line_number	Optional, if included, the command resets one breakpoint at this line. **line_number** can be a line number or a label.

Comments

The **buffer_number** qualifier in the command must specify one buffer only.

If **line_number** is omitted, the command resets all breakpoints in the buffer.

If the buffer qualifier is specified as #, for example, **##BR**, and **line_number** is omitted, the command resets all breakpoints in all buffers.

Example

`#0L`	List buffer 0
`1: MovePTP:` **`2: VEL(0) = 20000`** **`3: ptp  X, 4000`** **`4: till ^MST(0).#MOVE`** **`5: stop`**	Response
`#0BS3`	Set breakpoint at line 3
`#0L`	List buffer 0 (note the asterisk indicating the breakpoint)
`1: MovePTP:` **`2: VEL(0) = 20000`** **`3: *ptp  X, 4000`** **`4: till ^MST(0).#MOVE`** **`5: stop`**	Response
`#0XD`	Execute the program in debug mode in buffer 0
`?0`	Query buffer 0 state
`Buffer 0: 5 lines, suspended in line 3`	Response
`#0XS`	Execute line 3 of buffer 0
`?0`	Query buffer 0 state
`Buffer 0: 5 lines, suspended in line 4`	Response
`#0XD`	Execute the rest of the program in buffer 0

```
?0                                      Query buffer 0 state
Buffer 0: 5 lines, terminated in line 5 Response
```

5.4 System Commands

System commands provide you with information contained in the system.

5.4.1 SI

Description
The **SI** (System Information) command returns System Information about the SPiiPlus controller including serial number, firmware version, configuration, name and SP programs.

Syntax
#SI

Arguments
None

Example

```
#SI
Product:              ACTUAL2
Product Config#:      ACTUAL2
Product Serial#:      1234567
Firmware version:     0.84.15.03
Serial Number:        NT000000A
Part Number:          NT-LT

Hardware:
  MPU board:                  Nexcom EBC220-500MHz
  MPU board ID:               5
  Network Adapter:            RealTek RTL8139 PCI Ethernet card
  ID:                         3
  IP Adress:                  10.0.0.140
  MAC Address:                00 10 F3 0D B2 23
  Servo Processors:           0xSPii

Axes:
  Dummy:                      none
  DC Brush:                   0,1,2,3,4,5,6,7,12,13,14,15,16,17
  DC Brushless:               0,1,2,3,4,5,6,7,12,13,14,15,16,17
  P/D Stepper:                8,9,10,11
  LDM3U drives:               none
  PWM drives:                 0,1,2,3,4,5,6,7,12,13,14
  Digital Current Loop:       0,1,2,3,4,5,6,7,12,13,14
  Integrated drives:          0,1,2,3,4,5,6,7,12,13,14
                  Axis (0):    0.0A continuous/0.0A peak
                  Axis (1):    0.0A continuous/0.0A peak
                  Axis (2):    0.0A continuous/0.0A peak
                  Axis (3):    0.0A continuous/0.0A peak
                  Axis (4):    4.0A continuous/5.0A peak
                  Axis (5):    4.0A continuous/5.0A peak
                  Axis (6):    4.0A continuous/5.0A peak
```

```
                  Axis (7):    4.0A continuous/5.0A peak
                  Axis (12):    3.0A continuous/6.0A peak
                  Axis (13):    3.0A continuous/6.0A peak
                  Axis (14):    3.0A continuous/6.0A peak
  Remote HSSI drives:         none
  Dual loop:                  0,1,2,3,4,5,6,7,12,13,14,15,16,17
  PEG pulse:                  0,1,4,5,12,13,16,17
  PEG states:                 none

Options:
  SIN-COS Encoders:           0
  Input Shaping:              No
  SPiiPlus PLC:               No
  Axes with Servo Features:   0
  Servo Features Mask:        0x0000
  3rd Party Servo Axes:       0
  3rd Party Stepper Axes:     0
  3rd Party Digital I/O:      0
  3rd Party Analog I/O:       0

Network Unit 0:
  Vendor ID:                  0x00000540
  Product ID:                 0x01020000
  FPGA ID:                    0xFFFFFFFF
  FPGA version:               0xFFFFFFFF
  S/N:                        0
  Revision:                   0

  Network Axes:               0,1,2,3,4,5,6,7

  Inputs/Outputs:
     Digital inputs (IN):     0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7
     Digital outputs (OUT):   0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7
     Analog inputs (AIN):     0,1,2,3
     Analog outputs (AOUT):   0,1,2,3
     HSSI channels:           4
     Ext. inputs (EXTIN):     0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
     Ext. outputs (EXTOUT):   0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

  Integrated Component SP-DC-LT-8:
     Type:                    Controller (11)
     Address:                 0x007
     Subsystems:              2
     Production date:         15/08/10
     HW revision:             8
     S/N:                     88

  Integrated Component PS-48-07:
     Type:                    Power supply (new)
     Address:                 0x005
     Subsystems:              1
     Voltage:                 48V - 48V
     Power:                   700W
     Production date:         15/08/10
     HW revision:             23
     S/N:                     2345
```

```
  Integrated Component DD-4-320-3_1:
     Type:                   PWM drive (5)
     Address:                0x003
     Subsystems:             4
     Axes:                   0,1,2,3
         Drive 1:  Axis 0
         Drive 2:  Axis 1
         Drive 0:  Axis 2
         Drive 3:  Axis 3
     Voltage:                320V - 320V
     Nominal current:        3.000000A
     Peak current:           6.000000A
     RMS protection Time:    3476.000000
     Production date:        August 15, 2010
     HW revision:            6
     S/N:                    66

  Integrated Component DD-4-60-5:
     Type:                   PWM drive (5)
     Address:                0x002
     Subsystems:             4
     Axes:                   4,5,6,7
         Drive 0:  Axis 4
         Drive 2:  Axis 5
         Drive 1:  Axis 6
         Drive 3:  Axis 7
     Voltage:                60V - 60V
     Nominal current:        4.000000A
     Peak current:           5.000000A
     RMS protection Time:    3476.000000
     Production date:        21/06/10
     HW revision:            7
     S/N:                    89

Network Unit 2:
  Vendor ID:                 0x00000540
  Product ID:                0x02040000
  FPGA ID:                   0xFFFFFFFF
  FPGA version:              0xFFFFFFFF
  S/N:                       0
  Revision:                  0

  Network Axes:              8,9

  Inputs/Outputs:
     Digital inputs (IN):    1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7
     Digital outputs (OUT):  1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7
     Analog inputs (AIN):    none
     Analog outputs (AOUT):  none
     HSSI channels:          0
     Ext. inputs (EXTIN):    none
     Ext. outputs (EXTOUT):  none

  Integrated Component PD-NT-4-8-8:
     Type:                   SPiiPlus PDMnt
```

```
     Address:                0x207
     Subsystems:             1
     Production date:        01/01/10
     HW revision:            0
     S/N:                    0

Network Unit 3:
  Vendor ID:                 0x00000540
  Product ID:                0x01020000
  FPGA ID:                   0xFFFFFFFF
  FPGA version:              0xFFFFFFFF
  S/N:                       0
  Revision:                  0

  Network Axes:              10,11,12,13,14,15,16,17

  Inputs/Outputs:
     Digital inputs (IN):    2.0,2.1,2.2,2.3,2.4,2.5,2.6,2.7
     Digital outputs (OUT):  2.0,2.1,2.2,2.3,2.4,2.5,2.6,2.7
     Analog inputs (AIN):    4,5,6,7
     Analog outputs (AOUT):  4,5,6,7
     HSSI channels:          4
     Ext. inputs (EXTIN):    16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
     Ext. outputs (EXTOUT):  16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31

  Integrated Component SP-DC-LT-8:
     Type:                   Controller (11)
     Address:                0x307
     Subsystems:             2
     Production date:        21/06/10
     HW revision:            8
     S/N:                    888

  Integrated Component PS-48-07:
     Type:                   Power supply (new)
     Address:                0x305
     Subsystems:             1
     Voltage:                48V - 48V
     Power:                  700W
     Production date:        28/04/10
     HW revision:            1
     S/N:                    2

  Integrated Component DD-4-320-3:
     Type:                   PWM drive (5)
     Address:                0x303
     Subsystems:             4
     Axes:                   12,13,14
         Drive 0:  Axis 12
         Drive 2:  Axis 13
         Drive 1:  Axis 14
     Voltage:                320V - 320V
     Nominal current:        3.000000A
     Peak current:           6.000000A
     RMS protection Time:    3476.000000
     Production date:        25/08/10
```

```
     HW revision:           1
     S/N:                   1

SP0 Program Info:
  Monitor version:1
  Creation Date: Sun Jun 13 13:38:20 2010
  Saving Tool: SPiiPlus NT Servo Application File Generator v.6.83.07.00
  SPiiPlus NT Servo Processor Program.

  Date=     June 14th 2010
  Version=  1.0
  Firmware= 1.0

  ACS Motion Control Ltd.,
  Control and Applications Development,
  Copyright (c) 2010. All Rights Reserved.

SP1 Program Info:
  Monitor version:1
  Creation Date: Sun Jun 13 13:38:20 2010
  Saving Tool: SPiiPlus NT Servo Application File Generator v.6.83.07.00
  SPiiPlus NT Servo Processor Program.

  Date=     June 14th 2010
  Version=  1.0
  Firmware= 1.0

  ACS Motion Control Ltd.,
  Control and Applications Development,
  Copyright (c) 2010. All Rights Reserved.

SP2 Program Info:
  Monitor version:ffffffff
  Default Servo Processor Info.

SP3 Program Info:
  Monitor version:1
  Creation Date: Sun Jun 13 13:38:20 2010
  Saving Tool: SPiiPlus NT Servo Application File Generator v.6.83.07.00
  SPiiPlus NT Servo Processor Program.

  Date=     June 14th 2010
  Version=  1.0
  Firmware= 1.0

  ACS Motion Control Ltd.,
  Control and Applications Development,
  Copyright (c) 2010. All Rights Reserved.

SP4 Program Info:
  Monitor version:1
  Creation Date: Sun Jun 13 13:38:20 2010
  Saving Tool: SPiiPlus NT Servo Application File Generator v.6.83.07.00
  SPiiPlus NT Servo Processor Program.

  Date=     June 14th 2010
```

```
  Version=  1.0
  Firmware= 1.0

  ACS Motion Control Ltd.,
  Control and Applications Development,
  Copyright (c) 2010. All Rights Reserved.
```

5.4.2 IR

Description

The **IR** (Integrity Report) command activates integrity validation and provides a report of current integrity state. The report displays a list of files. Each list entry displays a file name, expected file size and checksum of the file and actual file size and checksum.

Note	
	It takes some time to compile the Integrity Report, in order to avoid a Timeout error: 1. *Right-click the controller in the Workspace Tree and select* **Properties**. 2. *Increase the* **Connection Timeout** *to 10,000 ms.* 3. *Click* **Connect**.

Syntax

#IR

Arguments

None

Comments

If any integrity problem is detected, the command raises fault bit **S_FAULT.#INTGR**.

Example

```
#IR
                    Size                    Checksum
                    Registered  Actual      Registered  Actual
c:\
  SB1218PC.frm      001E2870    001E2870    03672ED6    03672ED6
  SB1218PC.bin      0000812A    0000812A    DDC2E529    DDC2E529
c:\sb4\dsp\
  dsp.###           0004FF18    0004FF18    61F6BA11    61F6BA11
  dsp.##1           0003E11B    0003E11B    783AE65B    783AE65B
  adj0.$$$          0000122B    0000122B    69003B3B    69003B3B
  adj1.$$$          00001A99    00001A99    055D4E11    055D4E11
  adj2.$$$          000019DA    000019DA    BC928A33    BC928A33
  adj3.$$$          00001A9B    00001A9B    4403628F    4403628F
  adj4.$$$          0000197E    0000197E    DE12BDD7    DE12BDD7
  adj5.$$$          00001229    00001229    A974E209    A974E209
  adj6.$$$          00001229    00001229    A977E20C    A977E20C
  adj7.$$$          0000122B    0000122B    61EE382F    61EE382F
c:\sb4\startup\
  par.$$$           000001C9    000001C9    D8A0220A    D8A0220A
  par0.$$$          00000A16    00000A16    B2783961    B2783961
```

```
  par1.$$$          00000A13    00000A13    5E2EAD92    5E2EAD92
  par2.$$$          00000A14    00000A14    5A972E6D    5A972E6D
  par3.$$$          00000A13    00000A13    B25EDAD0    B25EDAD0
  par4.$$$          00000A13    00000A13    CD6E01E2    CD6E01E2
  par5.$$$          00000A13    00000A13    F6873205    F6873205
  par6.$$$          00000A15    00000A15    39E27520    39E27520
  par7.$$$          00000A15    00000A15    77CA503D    77CA503D
  par8.$$$          00000A13    00000A13    43A67043    43A67043
  par9.$$$          00000A13    00000A13    67BC915E    67BC915E
  par10.$$$         00000A89    00000A89    454F04F9    454F04F9
  par11.$$$         00000A89    00000A89    5A67281F    5A67281F
  par12.$$$         00000A89    00000A89    6F7F4B45    6F7F4B45
  par13.$$$         00000A89    00000A89    84976E6B    84976E6B
  par14.$$$         00000A89    00000A89    99AF9191    99AF9191
  par15.$$$         00000A89    00000A89    AEC7B4B7    AEC7B4B7
  par16.$$$         00000A89    00000A89    C3DFD7DD    C3DFD7DD
  par17.$$$         00000A89    00000A89    D8F7FB03    D8F7FB03
  par18.$$$         00000A89    00000A89    EE101E29    EE101E29
  par19.$$$         00000A89    00000A89    0328414F    0328414F
  par20.$$$         00000A89    00000A89    6B641D1C    6B641D1C
  par21.$$$         00000A89    00000A89    807C4042    807C4042
  par22.$$$         00000A89    00000A89    95946368    95946368
  par23.$$$         00000A89    00000A89    AAAC868E    AAAC868E
  par24.$$$         00000DF1    00000DF1    26E26061    26E26061
  par25.$$$         00000DF1    00000DF1    4B01777C    4B01777C
  par26.$$$         00000DF1    00000DF1    6F208E97    6F208E97
  par27.$$$         00000DF1    00000DF1    933FA5B2    933FA5B2
  par28.$$$         00000DF1    00000DF1    B75EBCCD    B75EBCCD
  par29.$$$         00000DF1    00000DF1    DB7DD3E8    DB7DD3E8
  par30.$$$         00000DF1    00000DF1    B18A230C    B18A230C
  par31.$$$         00000DF1    00000DF1    D5A93A27    D5A93A27
c:\sb4\user\
  I                 000001A0    000001A0    414453BB    414453BB
  V                 00000330    00000330    414453BC    414453BC
  X_FILE            00000070    00000070    82665362    82665362
  PMAP              00000340    00000340    F29BADD0    F29BADD0
  ONE               00000FC0    00000FC0    4144573F    4144573F
  TWO               00002F00    00002F00    414453A7    414453A7
  X_ERR             00000040    00000040    42A2135C    42A2135C
System Integrity is OK
```

5.4.3 U

Description
The **U** (Usage) command is used for monitoring MPU usage. It returns the maximum, average, and minimum values as a percent.

Syntax
#U

Arguments
None

Comments
The controller continuously measures the time taken by real-time tasks. When the **U** command is received, the controller analyzes the measured times during the last 50 controller cycles and calculates minimal, maximal and average time. The results are reported in percents.

5.4.4 TD

Description
The **TD** command returns the names of all user-defined variables and arrays that are in the controller flash memory.

Synax
#TD

Arguments
None

5.4.5 SC

Description
The **SC** (Safety Control) command reports the current safety system configuration.

The controller response includes the following:

- ❑ active safety groups
- ❑ the configuration of each fault for each motor

Syntax
#SC

Arguments
None

Example
#SC

```
 Bit  Code    Fault                0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
  0  #RL     Right Limit          K   K   K   K   K   K   K   K   K   K   K   K   K   K   K
  1  #LL     Left Limit           K   K   K   K   K   K   K   K   K   K   K   K   K   K   K
  2  #NT     Network error        D   D   D   D   D   D   D   D   D   D   D   D   D   D   D
  3  #FAN    Cooling Fan Fault     -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
  4  #HOT    Overheat             -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
  5  #SRL    Software Right Limit   K   K   K   K   K   K   K   K   K   K   K   K   K   K   K
  6  #SLL    Software Left Limit    K   K   K   K   K   K   K   K   K   K   K   K   K   K   K
```

```
  7 #ENCNC   Encoder Not Connected   D   D   D   D   D   D   D   D   D   D   D   D   D   D   D   D
  8 #ENC2NC  Encoder 2 Not Connected -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
  9 #DRIVE   Driver Alarm           D   D   D   D   D   D   D   D   D   D   D   D   D   D   D   D
 10 #ENC     Encoder Error          D   D   D   D   D   D   D   D   D   D   D   D   D   D   D   D
 11 #ENC2    Encoder 2 Error        -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 12 #PE      Position Error
 13 #CPE     Critical Position Error D   D   D   D   D   D   D   D   D   D   D   D   D   D   D   D
 14 #VL      Velocity Limit         K   K   K   K   K   K   K   K   K   K   K   K   K   K   K   K
 15 #AL      Acceleration Limit      -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 16 #CL      Overcurrent           D   D   D   D   D   D   D   D   D   D   D   D   D   D   D   D
 17 #SP      Servo Processor Alarm   D   D   D   D   D   D   D   D   D   D   D   D   D   D   D   D
 20 #HSSINC  HSSI Not Connected      -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 25 #PROG    Program Error          -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 26 #MEM     Memory Overuse         -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 27 #TIME    Time Overuse           -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 28 #ES      Emergency Stop         D   D   D   D   D   D   D   D   D   D   D   D   D   D   D   D
 29 #INT     Servo Interrupt        -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 30 #INTGR   Integrity Violation     -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 31 #FAILURE Component Failure       -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
Legend:
  -      Fault detection disabled
  Blank  No default response
  K      Kill response
  D      Disable response
  KD     Kill-disable response
  +      Generalized fault
 Bit  Code     Fault                   16   17
   0  #RL      Right Limit              K    K
   1  #LL      Left Limit               K    K
   2  #NT      Network error            D    D
   3  #FAN     Cooling Fan Fault        -    -
   4  #HOT     Overheat                 -    -
   5  #SRL     Software Right Limit     K    K
   6  #SLL     Software Left Limit      K    K
   7  #ENCNC   Encoder Not Connected    D    D
   8  #ENC2NC  Encoder 2 Not Connected  -    -
   9  #DRIVE   Driver Alarm             D    D
  10  #ENC     Encoder Error            D    D
  11  #ENC2    Encoder 2 Error          -    -
  12  #PE      Position Error
  13  #CPE     Critical Position Error  D    D
  14  #VL      Velocity Limit           K    K
  15  #AL      Acceleration Limit       -    -
  16  #CL      Overcurrent              D    D
  17  #SP      Servo Processor Alarm    D    D
  20  #HSSINC  HSSI Not Connected       -    -
  25  #PROG    Program Error            -    -
  26  #MEM     Memory Overuse           -    -
  27  #TIME    Time Overuse             -    -
  28  #ES      Emergency Stop           D    D
  29  #INT     Servo Interrupt          -    -
  30  #INTGR   Integrity Violation      -    -
  31  #FAILURE Component Failure        -    -
Legend:
  -      Fault detection disabled
  Blank  No default response
  K      Kill response
  D      Disable response
  KD     Kill-disable response
  +      Generalized fault
:
The following designations are used in the report:
--- – fault detection is disabled (FMASK=0)
(blank) – fault response is disabled (FDEF=0) or no default response is defined
K – response is kill
D – response is disable
KD – response is kill-disable
+ – generalized fault
```

5.4.6 ETHERCAT

Description
The **ETHERCAT**command is used for obtaining complete information about the connected EtherCAT slaves.

The information it displays is:

- Slave number
- Vendor ID
- Product ID
- Revision
- Serial number
- EtherCAT physical address
- DC support
- Mailbox support

Afterwards the list of network variables is listed. Each variable is described with:

- Name (as in XML)
- Offset inside the telegram (magic number that is used for mapping)
- IN or OUT description
- Data size

Syntax
#ETHERCAT

Arguments
None

Example
```
#ETHERCAT

EtherCAT bus scan found 2 slaves:
Slave 0:
========
Name:        Box 1 (MCPlus)
Vandor ID:   0x00000540    Product ID:   0x01020000
PHYS ADDR:   0x03E9   Alias:  0x0000
PD IN:       Offset   26.0 Size  118
PD OUT:      Offset   26.0 Size  136
STATE:       OP
Slave 1:
========
Name:        Box 2 (MCPlus)
Vandor ID:   0x00000540    Product ID:   0x01020000
PHYS ADDR:   0x03EA   Alias:  0x0000
PD IN:       Offset  162.0 Size  118
PD OUT:      Offset  162.0 Size  136
```

```
STATE:       OP


Network variables:
==================
Offset  Size  Dir   Name
  26     32   In    Command response1
  30     32   In    Command response2
  34     32   In    Command response3
  38     32   In    Command response4
  42     32   In    Command response5
  46     32   In    Command response6
  50     32   In    Command response7
  54     32   In    Command response8
  58     32   In    Feedback Position1
  62     32   In    Reference Position1
  66     32   In    Drive status1
  70     32   In    GP data 1
  74     32   In    Feedback Position2
  78     32   In    Reference Position2
  82     32   In    Drive status2
  86     32   In    GP data 2
  90     32   In    Feedback Position3
  94     32   In    Reference Position3
  98     32   In    Drive status3
 102     32   In    GP data 3
 106     32   In    Feedback Position4
 110     32   In    Reference Position4
 114     32   In    Drive status4
 118     32   In    GP data 4
 122     16   In    Digital inputs
 124     16   In    FPGA status
 126     16   In    HSSI 1
 128     16   In    HSSI 2
 130     16   In    HSSI 3
 132     16   In    HSSI 4
 134     16   In    HSSI 5
 136     16   In    HSSI 6
 138     16   In    HSSI 7
 140     16   In    HSSI 8
 142     16   In    Sync Counter
 162     32   In    Command response1
 166     32   In    Command response2
 170     32   In    Command response3
 174     32   In    Command response4
 178     32   In    Command response5
 182     32   In    Command response6
 186     32   In    Command response7
 190     32   In    Command response8
 194     32   In    Feedback Position1
 198     32   In    Reference Position1
 202     32   In    Drive status1
 206     32   In    GP data 1
 210     32   In    Feedback Position2
 214     32   In    Reference Position2
 218     32   In    Drive status2
```

```
222     32   In    GP data 2
226     32   In    Feedback Position3
230     32   In    Reference Position3
234     32   In    Drive status3
238     32   In    GP data 3
242     32   In    Feedback Position4
246     32   In    Reference Position4
250     32   In    Drive status4
254     32   In    GP data 4
258     16   In    Digital inputs
260     16   In    FPGA status
262     16   In    HSSI 1
264     16   In    HSSI 2
266     16   In    HSSI 3
268     16   In    HSSI 4
270     16   In    HSSI 5
272     16   In    HSSI 6
274     16   In    HSSI 7
276     16   In    HSSI 8
278     16   In    Sync Counter
310     32   In    DC1
314     32   In    DC2
318     32   In    DC3
322     32   In    DC4
326     32   In    DC5
330     32   In    DC6
334     32   In    DC7
338     32   In    DC8
342     32   In    DC9
346     32   In    DC10
350     32   In    DC11
354     32   In    DC12
358     32   In    DC13
362     32   In    DC14
366     32   In    DC15
370     32   In    DC16
374     32   In    DC17
378     32   In    DC18
382     32   In    DC19
386     32   In    DC20
390     32   In    DC21
394     32   In    DC22
398     32   In    DC23
402     32   In    DC24
406     32   In    DC25
410     32   In    DC26
414     32   In    DC27
418     32   In    DC28
422     32   In    DC29
426     32   In    DC30
430     32   In    DC31
434     32   In    DC32
438     32   In    DC33
442     32   In    DC34
446     32   In    DC35
450     32   In    DC36
```

```
454     32   In    DC37
458     32   In    DC38
462     32   In    DC39
466     32   In    DC40
482     32   In    DC1
486     32   In    DC2
490     32   In    DC3
494     32   In    DC4
498     32   In    DC5
502     32   In    DC6
506     32   In    DC7
510     32   In    DC8
514     32   In    DC9
518     32   In    DC10
522     32   In    DC11
526     32   In    DC12
530     32   In    DC13
534     32   In    DC14
538     32   In    DC15
542     32   In    DC16
546     32   In    DC17
550     32   In    DC18
554     32   In    DC19
558     32   In    DC20
562     32   In    DC21
566     32   In    DC22
570     32   In    DC23
574     32   In    DC24
578     32   In    DC25
582     32   In    DC26
586     32   In    DC27
590     32   In    DC28
594     32   In    DC29
598     32   In    DC30
602     32   In    DC31
606     32   In    DC32
610     32   In    DC33
614     32   In    DC34
618     32   In    DC35
622     32   In    DC36
626     32   In    DC37
630     32   In    DC38
634     32   In    DC39
638     32   In    DC40
 26     16   Out   Command1
 28     16   Out   Command2
 30     16   Out   Command3
 32     16   Out   Command4
 34     16   Out   Command5
 36     16   Out   Command6
 38     16   Out   Command7
 40     16   Out   Command8
 42     32   Out   Command Arg1
 46     32   Out   Command Arg2
 50     32   Out   Command Arg3
 54     32   Out   Command Arg4
```

```
 58     32   Out   Command Arg5
 62     32   Out   Command Arg6
 66     32   Out   Command Arg7
 70     32   Out   Command Arg8
 74     32   Out   Direct Command1
 78     32   Out   Reference Acceleration1
 82     32   Out   Reference Velocity1
 86     32   Out   Reference Position1
 90     32   Out   Controller status1
 94     32   Out   Direct Command2
 98     32   Out   Reference Acceleration2
102     32   Out   Reference Velocity2
106     32   Out   Reference Position2
110     32   Out   Controller status2
114     32   Out   Direct Command3
118     32   Out   Reference Acceleration3
122     32   Out   Reference Velocity3
126     32   Out   Reference Position3
130     32   Out   Controller status3
134     32   Out   Direct Command4
138     32   Out   Reference Acceleration4
142     32   Out   Reference Velocity4
146     32   Out   Reference Position4
150     32   Out   Controller status4
154     16   Out   analog output1
156     16   Out   analog output2
158     16   Out   digital output
160     16   Out   Sync Counter
162     16   Out   Command1
164     16   Out   Command2
166     16   Out   Command3
168     16   Out   Command4
170     16   Out   Command5
172     16   Out   Command6
174     16   Out   Command7
176     16   Out   Command8
178     32   Out   Command Arg1
182     32   Out   Command Arg2
186     32   Out   Command Arg3
190     32   Out   Command Arg4
194     32   Out   Command Arg5
198     32   Out   Command Arg6
202     32   Out   Command Arg7
206     32   Out   Command Arg8
210     32   Out   Direct Command1
214     32   Out   Reference Acceleration1
218     32   Out   Reference Velocity1
222     32   Out   Reference Position1
226     32   Out   Controller status1
230     32   Out   Direct Command2
234     32   Out   Reference Acceleration2
238     32   Out   Reference Velocity2
242     32   Out   Reference Position2
246     32   Out   Controller status2
250     32   Out   Direct Command3
254     32   Out   Reference Acceleration3
```

```
 258     32   Out   Reference Velocity3
 262     32   Out   Reference Position3
 266     32   Out   Controller status3
 270     32   Out   Direct Command4
 274     32   Out   Reference Acceleration4
 278     32   Out   Reference Velocity4
 282     32   Out   Reference Position4
 286     32   Out   Controller status4
 290     16   Out   analog output1
 292     16   Out   analog output2
 294     16   Out   digital output
 296     16   Out   Sync Counter
```

5.4.7 CC

Description

The CC returns data on the existing communication channels.

Syntax

#CC

Example

```
#CC
Channel Type      Mode
 1      Serial    Command  Rate:115200 0N 1
 2      Serial    Command  Rate:115200 0N 1
 6      TCP/IP    (  701)  Peer:None
 7      TCP/IP    (  701)  Peer:10.0.0.52
 8      TCP/IP    (  701)  Peer:10.0.0.65
 9      TCP/IP    (  701)  Peer:10.0.0.58
10      UDP       (  700)  Peer:N/A
36      TCP/IP    (  701)  Peer:None
37      TCP/IP    (  701)  Peer:None
38      TCP/IP    (  701)  Peer:None
39      TCP/IP    (  701)  Peer:None
12      PCI bus   Command
16      TCP/IP    MODBUS Slave  Connection:network  Peer:None
```

6 SPiiPlus Error Codes

This chapter contains explanations of the Error Codes that may appear.

The chapter contains only those Error Codes returned by the controller and does not include errors associated with the C Library. Errors that are detected by the C Library are never returned by the controller. A host application that calls a C Library function can receive these codes if a function call failed. For explanations of the C Library errors see the *SPiiPlus C Library Reference Guide*.

The ACSPL+ Error Codes range 1000 to 6011 and are broken down as follows:

ACSPL+ Syntax Errors – numbers 1000 to 1999

ACSPL+ Compilation Errors – numbers 2000 to 2999

ACSPL+ Runtime Errors – numbers 3000 to 3999

Motion Termination Errors – numbers 5000 to 5150

System Errors - numbers 5151 to 5999

EtherCAT Errors - numbers 6000 to 6999

Note	
	Error code values of 7000 through 8999 are reserved for future use and are currently not used. *Error code values of 9000 and above are reserved for user-defined error codes.*

6.1 ACSPL+ Syntax Errors

These appear in response to syntax errors that the controller detects when a program is entered into the buffer and are reported immediately in the prompt that is displayed in response to the command.

The error code values range between 1000 and 1999.

Table 63 ACSPL+ Syntax Errors (page 1 of 17)

Error Code	Error Message	Remarks
1001	The program is suspended	The program is being run in the Step mode, and has been suspended at the current step.
1002	The program was terminated by user	The program has reached a user-set breakpoint.
1020	Illegal subcommand	A subroutine command that is not recognized has been entered.

Table 63 ACSPL+ Syntax Errors (page 2 of 17)

Error Code	Error Message	Remarks
1021	SP command requires axis specification	The required axis designation is missing.
1022	Illegal command	A program command that is not recognized has been entered.
1023	Read-only variable cannot be assigned	The command specifies assignment to a read-only variable.
1024	Set variable cannot be reported	
1025	Time interval between two characters in command is more than 2 seconds	The compiler recognizes that a follow-on command has not arrived within 2 seconds. The error is relevant when an application is using a call to a SPiiPlus C Library routine, and usually indicates a communication problem.
1026	Serial Number, Part Number, or Software Options were already specified	An attempt was made to respecify the controller's serial number, part number, or software options.
1027	Variable requires axis specification	An ACSPL+ variable missing the required axis specification was used.
1028	Scalar variable cannot accept axis specification	A scalar variable was used with an axis specification.
1029	Extra characters after the command	A command has been entered with extraneous characters or parameters that are not recognized.
1030	Too many parameters	The entry contains too many parameters.
1031	Illegal array in the array command	The array in the command either does not exist, or its size does not match the requirements of the command.
1032	Illegal data in array	The array contains data in a format that does not match requirements.
1033	Illegal edit command	The edit command that has been entered cannot be executed.
1034	Illegal index value	The command includes numerical index specification, but the specified index is not a number.

Table 63 ACSPL+ Syntax Errors (page 3 of 17)

Error Code	Error Message	Remarks
1035	Index is out of range	The error is caused by one the following: ❑ The specified index value is more than the number of elements in the array ❑ The specified index value is negative ❑ The specified index values are incompatible (first value in the range greater than last).
1036	Internal error	An internal error has been detected..
1037	Illegal variable name	The command requires specification of an ACSPL+ variable name, but the specified name is not that of an ACSPL+ variable.
1038	Wrong checksum in the command	The command contains checksum, and the checksum value is wrong. The error is relevant when an application includes SPiiPlus C Library routines, and indicates communication problems.
1039	Only one motion per axis can be planned in advance	An attempt was made to setup more than one motion for a specified axis.
1040	Unable to open file	The command specifies an internal file in the flash memory that does not exist.
1041	Assigned value is out of range	The command attempts to assign an ACSPL+ variable with a value that is out of the range allowed for this variable.
1042	Operation failed because of exception	
1043	Program cannot start because the buffer was not compiled	The command attempts to start an un-compiled ACSPL+ program. To compile a program, in the SPiiPlus MMI Application Studio: ❑ In the **Program Manager**, right-click the buffer and select **Compile Buffer**, or ❑ Use the **#nC** command in the **Communication Terminal**, where **n** is the buffer number.

Table 63 ACSPL+ Syntax Errors (page 4 of 17)

Error Code	Error Message	Remarks
1044	Command cannot be executed while the program is running	The command attempts to affect a running ACSPL+ program. Stop the program before executing the command. To stop a program, in the SPiiPlus MMI Application Studio: ❑ In the **Program Manager**, right-click the buffer and select **Stop Buffer**, or ❑ Use the **#nS** command in the **Communication Terminal**, where **n** is the buffer number.
1045	Numerical error in standard function	The command includes an ACSPL+ function that caused a numerical error. Check if the arguments of the function fall into the allowed range.
1046	Write file error	The command has caused writing to the flash memory that failed. Possible reason is a flash memory overflow because of too many stored user files.
1047	Read file error	The command has caused reading from the flash memory that failed. A reoccurring error points to a serious problem in the controller hardware or firmware.
1048	More axes than were defined in the motion	The **POINT**command specifies more axes than were specified in the motion that the command refers to.
1049	Axis already belongs to a group	An attempt was made to assign an axis to a group when it has already been assigned to another group.
1050	Conflict with user-defined axis group	The command is incompatible with a user-defined axis group that was defined before. The axes specified in the command must either all belong to one user-defined axis group, or not to intersect with any user-defined axis group.
1051	Line number is out of range	The command specifies a line number that does not exist in the specified program buffer.
1052	Buffer number is out of range	The command specifies illegal buffer number. The controller contains 16 program buffers numbered from 0 to 15, plus the D buffer.

Table 63 ACSPL+ Syntax Errors (page 5 of 17)

Error Code	Error Message	Remarks
1053	Wrong type	The command addresses an ACSPL+ variable and the type of the variable is different from the type specified in the command. The error is relevant when an application includes a SPiiPlus C Library routine. The error indicates problems in communication.
1054	This type of motion is valid for single axis only	An attempt was made to send a single-axis motion to more than one axis.
1055	Command requires line number specification	The command is missing a required line number specification.
1056	Parameter MA defines illegal master value	The parameter specifying the master has an illegal value (see **MASTER**).
1057	Previous superimposed motion is in progress	
1058	Slave is not syncronized to master	The slave controller has not been syncronized with the master controller.
1059	Command PTPV must specify velocity value	The **PTP**/v command was issued without a value for velocity.
1060	Illegal memory command	The memory management command is improperly formatted.
1061	')' wasn't found	The command contains a non-paired left bracket.
1063	Variable is not defined in the buffer	The command addresses a variable that is not declared in the specified buffer, or the specified buffer is not compiled.
1064	Undefined global variable	The command addresses a global variable that is not declared in any buffer, or the buffer that contains the declaration is not compiled. Variable name must be in upper case.
1065	Command cannot be executed while the current motion is in progress	The command is in conflict with one or more of the currently executed motions. To kill a motion use **KILL** or **KILLALL**.
1067	GO command failed	The controller has no motions waiting for the **GO** command.

Table 63 ACSPL+ Syntax Errors (page 6 of 17)

Error Code	Error Message	Remarks
1068	Referenced axis does not execute a motion (motion was terminated?)	The command refers to a motion, but the specified axis is not involved in any motion.
1069	This command can be used only with MPTP, PATH or PVSPLINE motion	Commands **POINT**and **MPOINT**are compatible only with the **MPTP...ENDS**, **PATH...ENDS**, or **PVSPLINE...ENDS** motions.
1070	Attempt to add segment after ENDS command	The command attempts to add a segment or a point to the motion that was closed before with an **ENDS** of the **MPTP...ENDS**, **PATH...ENDS**, or **PVSPLINE...ENDS** commands.
1071	File name was expected	The command must specify a name of internal file in the flash memory.
1072	Wrong array size	The command specifies an array, but the motion to which the command refers, or other command arguments require an array of another size.
1073	Text for search is not specified	The command must specify a text for search operation.
1074	Only standard or SP variable is allowed in the command	The command requires an ASCPL+ or SP variable name to be specified.
1075	Name is not a standard or user variable	The specified variable name is not an ACSPL+ or user-defined variable name.
1076	Undefined label	The command requires a label specification. The program that contains the label specified in the command must be compiled in a buffer.
1077	Protection violation	The command attempts to assign a protected variable when the controller is in Protected mode. The controller must be put in the Not Protected mode before protected variables can be assigned. To do this, use the SPiiPlus MMI Application Studio **Protection Wizard** in **Development Tools**.

Table 63 ACSPL+ Syntax Errors (page 7 of 17)

Error Code	Error Message	Remarks
1078	Variable can be changed only while the motor is disabled	The command attempts to assign a value to a variable that can be changed only if the motor is disabled. A **DISABLE** command must be put in before the value can be assigned to the variable.
1079	Motion cannot start because one or more motors are disabled	Motor(s) are in motion and have to be disabled before the motion can be initiated. A **DISABLE** command must be put in prior to initiating motion.
1080	Default Connection flag is set for the motor	
1081	Incompatible suffixes	The command includes command options that cannot be used together.
1082	Commands BEGIN, END, KILL, GO require axis specification	A **GO** or **KILL** command has been entered without an axis specification.
1083	Array requires axis specification	An array requiring an axis specification has been entered without an axis specification.
1084	Illegal array command	Either the array does not exist, or it is not in a compiled buffer.
1085	Extra number after the command	The command specifies a superfluous numerical argument.
1086	Variable name must be specified	The command requires a variable name specification.
1087	Command cannot be executed while the axis is moving	The command specifies one or more axes that are involved in a motion. The command can be executed only after the motion finishes.
1088	Variable can be queried only in compiled buffer	The command attempts to query a variable in a buffer that was not compiled.
1089	Label can be referenced only in compiled buffer	The command attempts to reference a label in a buffer that was not compiled.
1090	This type of motion is not allowed for grouped axis	The slave or track command specifies an axis that was included in a user-defined group. Only a single axis can be specified in the slave or track command.

Table 63 ACSPL+ Syntax Errors (page 8 of 17)

Error Code	Error Message	Remarks
1091	Less arguments than required	The motion command specifies fewer coordinate values than required by the axis specification.
1092	More arguments than required	The motion command specifies more coordinate values than required by the axis specification.
1093	Bit selector value is greater than 31 or less than 0	The expression specified in the bit selector yields a value greater than 31 or less than 0.
1094	Empty line range is specified	The range of lines specified refers to empty lines.
1095	No master-slave motion is in progress	The command addresses master-slave motion that has not yet started.
1096	'}' was not found	The command includes a non-paired left curly bracket.
1097	Previous data collection is in progress	The command attempts to start a data collection while the previous data collection for the same axis is still in progress.
1098	Stalled motion requires limits specification	The motion command, which includes a command option of stalled motion, requires specification of the motion limits.
1099	Extra numbers after the command	The motion command includes superfluous numerical arguments.
1100	Received command has no message ID	The command does not contain the ID of the message to which it refers.
1101	The program is suspended, the start line number is not allowed	The command attempts to start a program from a specified line when the program is in suspended state. A program in suspended state can be only resumed from the next line. The line specification is not allowed. The only way to start a suspended program from arbitrary line is to stop the program, and to start it again from the desired line.
1102	Zero divide	The an illegal divide by zero is attempted by the command.
1103	Invalid reference	The command contains an invalid reference.

Table 63 ACSPL+ Syntax Errors (page 9 of 17)

Error Code	Error Message	Remarks
1104	No ACSPL program is waiting for input	The command is attempting to send input, but no program is open for receipt.
1105	Format error	The command is incorrectly formatted.
1106	SP function failed	The function that accesses SP memory cannot be executed. Check the address specified in the function call. The address must fall into the range of 0 to 511.
1107	Current empty space in the dynamic buffer is not enough for the command	The D-Buffer is full and no further items can be entered.
1108	Invalid real number	The command includes specification of a real number that cannot be interpreted as a valid real.
1109	The command is not allowed in SAFE mode	Safe communication mode is a host communication format based on block-by-block transfer of data with delivery confirmation.
1110	At least one variable must be specified for data collection	Data collection was attempted without specifying at least one variable.
1111	Too long reply requested	The time of the reply request parameter is too long.
1112	No matches were found	The search command did not find any match.
1113	The step in the table is zero or negative	The command reference to a table element is either zero or negative.
1114	The program finished without STOP command	The program that runs in a buffer executed the last command and this was not a **STOP** command.
1115	Stack underflow (RET without CALL)	The program that runs in a buffer caused stack underflow. This occurs if the program executes the **RET** command that without the paired **CALL** command.

Table 63 ACSPL+ Syntax Errors (page 10 of 17)

Error Code	Error Message	Remarks
1116	Stack overflow (too many nested CALLs)	The program that runs in a buffer caused stack overflow. This occurs if the program executes too many nested **CALL** commands. Check that the program does not specify infinite recursion (subroutine calls itself infinitely).
1117	Attempt to enter autoroutine directly	The program that runs in a buffer comes to the **ON** command (see **ON...RET**). **ON** must never be executed in the normal program flow.
1118	Illegal axis number	The command specifies an axis by number or expression, and the resulting value is not a legal axis number. Valid axis numbers range between 0 and the number of axes in the system minus one.
1119	Integer overflow	The integer value is too large.
1120	The motion must involve the first two axes from the group	The segmented motion must always involve two axes. If a user-defined group contains more than two axes, the segmented motion must involve the first two axes in the group.
1121	Unknown #-constant	The command specifies an unknown symbolic constant.
1122	Bit selection is not applicable	Bit selector cannot be applied to real value.
1123	Illegal bit selector	Value of bit selector must fall in the range of 0 to 31.
1124	Attempt to enable motor failed	The enable command failed. Additional information can be obtained from the corresponding element of the **MERR** variable (motor disable code).
1125	Error in SP program	The command caused unsuccessful loading of an SP program. The file with the SP program contains an error.
1126	Illegal SP number	The command specifies an illegal SP number. Valid SP numbers are 0, 1, 2, and 3.

Table 63 ACSPL+ Syntax Errors (page 11 of 17)

Error Code	Error Message	Remarks
1127	Editing of this buffer is disabled	The command attempts to open the buffer for editing or to change the program in a Protected buffer. Editing a buffer is disabled if the controller is in the Protected mode and bit 1 of the corresponding element of **PFLAGS** is set to 1.
1128	Configuration changed. Commands SAVE and HWRES must be executed.	The program has changed the configuration. The configuration has to be saved to the controller flash and the controller restarted. Select the controller in the Workspace Tree and click the **Save to Flash** button. Then right-click the controller and select **Reboot**.
1129	In binary command the name specification must end with /	The command syntax requires a name to be followed by the / (slash) character.
1130	Segment sequence for the previous motion were not terminated with ends command.	Commands **MPTP...ENDS**, **MSEG...ENDS**, **PATH...ENDS**, **PVSPLINE...ENDS** are followed by a sequence of points or segments. The sequence must terminate with **ENDS**.
1131	SP program with unknown interface: NORMAL is assumed	SP program interface is unrecognized.
1132	The file is not a legal user data file	The file designator is not valid.
1133	Discrepancy in types of the saved and restored arrays	Error detected in array types when comparing the saved and restored versions.
1134	Discrepancy in sizes of the saved and restored arrays	Error detected in array size when comparing the saved and restored versions.
1135	Operation failed because of communication overload	Error occurs when there is either a fault in the communication, or too much communication is being conducted.
1136	Wrong relation between first point, last point and interval	Usually caused by the first point being greater than the last point.
1137	Illegal analog output number	The command contains an illegal analog output number.
1138	Incompatible SP and analog output	Conflict between analog output value in the command and that contained in the SP.

Table 63 ACSPL+ Syntax Errors (page 12 of 17)

Error Code	Error Message	Remarks
1139	Illegal input	The controller tries to interpret the inserted string as a response to the executed input command, but the string does not follow the required format.
1140	The function is not supported	An attempt was made to use a function that is not supported in SPiiPlus NT.
1141	Timeout	Communication timeout has occurred. This can be corrected by right-clicking the controller in the Workspace Tree, selecting **Properties**, and increasing the **Connection Timeout** value.
1142	Arguments are inconsistent	Conflict with the command arguments and their values.
1143	Memory overflow	Usually caused by infinite loops.
1144	Simulator does not support this command	The command cannot be executed by Simulator.
1145	The specified DPR address is less than 128 or exceeds the DPR size	SPiiPlus NT does not contain a dual-port RAM; therefore this is not relevant.
1146	Collision with other variable in DPR	SPiiPlus NT does not contain a dual-port RAM; therefore this is not relevant.
1147	Incomplete command (intrusion of other process?)	The command is not correctly formulated.
1148	Requested more SINCOS encoder multipliers than installed	The user tried to select more Sin-Cos multipliers than allowed.
1149	Illegal SP address	Valid SP numbers are 0, 1, 2, and 3.
1150	Only even numbers are allowed as DPR address	SPiiPlus NT does not contain a dual-port RAM. If the program has been imported from an older non-NT version and this error appears, delete any DPR command (such as **COPYFROMDPR** or **COPYTODPR**).
1151	This is a DEMO version of Simulator. 5 minutes of work remains.	The demo version of the Simulator has a limited session length. The Simulator is going to stop after 5 minutes.
1152	The DEMO version of Simulator has terminated	The demo version of the Simulator has a limited session length. The session time has elapsed.

Table 63 ACSPL+ Syntax Errors (page 13 of 17)

Error Code	Error Message	Remarks
1153	Illegal query command	The controller cannot recognize the query command.
1154	This command can be used only with MSEG motion	The command was incorrectly used - it can only be used with **MSEG...ENDS**.
1155	Motion cannot start because the previous motion failed. Use FCLEAR command.	To start the motion enter the **FCLEAR** command.
1156	Profile key must be specified as /SECTION/KEY/	
1157	Illegal configuration string. Use only characters K,D,+,-.	Illegal characters were used in the configuration string.
1158	Cannot find matching value. The formula has no root or the root is not single.	
1159	Axis number is specified more than once	The axis designation parameter is repeated in the command.
1160	The axis cannot be used in a group (see AFLAGS)	The specified axis has been flagged as not to be grouped using **AFLAGS**.
1161	Illegal communication channel	An invalid communication channel has been specified.
1162	Illegal tag value	The command Tag parameter does not match the intended command.
1163	Illegal configuration parameters	Invalid system configuration parameters have been entered.
1164	Illegal password	The password that has been entered does not match the required password.
1165	Attempt to execute optional function which is not installed	
1166	Flash file operation failed (overlapped file operations?)	
1167	Wrong start position is specified	
1168	File not opened	

Table 63 ACSPL+ Syntax Errors (page 14 of 17)

Error Code	Error Message	Remarks
1169	D-Buffer cannot be changed while any other buffer is compiled	An attempt was made to make changes to the D-Buffer while other buffers were being compiled.
1171	The number of axes in coordinated motion is restricted to 4	
1172	Illegal category name	
1173	EtherCAT offset is out of range	
1174	Can't find EtherCAT network variable	
1175	EtherCAT Master is not ready	
1176	Data size should be specified	
1177	This slave has no mailbox support	
1178	Invalid CoE SDO parameter	
1179	EtherCAT slave is in invalid state for this operation	
1180	General EtherCAT error	
1181	EtherCAT Timeout error	
1182	Can't split non-existing axes group	An axes group that was not previously defined was specified for splitting.
1183	Attempt to split group with active motion on	Cannot split an axis group while in motion.
1201	End-of-Sequence is illegal for this motion	
1211	Too many segments	See **MSEG...ENDS**.
1212	ARC arguments are inconsistent	See **MSEG...ENDS**, **ARC1** and **ARC2**.
1213	Stopper is prohibited for master-slave motion	See **STOPPER**.
1214	Adjacent stoppers are prohibited	See **STOPPER**.
1215	In cyclic path the first and the last points must coincide	

Table 63 ACSPL+ Syntax Errors (page 15 of 17)

Error Code	Error Message	Remarks
1216	Velocity is specified, but the motion command had no V suffix	Velocity argument can be specified in **POINT**, **LINE**, **ARC1** or **ARC2** only if the corresponding motion command specifies **/v**.
1217	Segment of zero length	
1221	End-of-Sequence is illegal for this motion	
1231	Illegal key	An invalid **Key** argument has been entered in either the **GETCONF** or **SETCONF** function.
1232	Invalid index	An invalid **Index** argument has been entered in either the **GETCONF** or **SETCONF** function.
1233	Invalid value	The **Value** argument of the **SETCONF** command specifies a value that is not compatible with the **Key** argument.
1234	Value in SETCONF must be 0 or 1	See **SETCONF**.
1235	SETCONF function is disabled	
1236	SETCONF cannot be executed while the motor is enabled	
1238	The operation can be executed only when the HSSI channel is in command mode	
1239	HSSI channel cannot switch to command mode because it affects one or more remote drivers. Reset corresponding MFLAGS.#HSSI bits before.	See **MFLAGS**.
1240	Operation is temporarily disabled	
1241	Operation failed	
1251	Operation is temporarily disabled	
1252	Motor cannot be enabled while a motion is in termination process	

Table 63 ACSPL+ Syntax Errors (page 16 of 17)

Error Code	Error Message	Remarks
1253	Unable to work with dummy motor	The command cannot be executed for a dummy motor. Check flag **MFLAGS.#DUMMY** (see **MFLAGS**).
1254	The operation requires the motor to be enabled	The **ENABLE/ENABLEALL** command has to be run.
1255	The operation requires the motor to be disabled	The **DISABLE** command has to be run.
1256	The operation is valid only for brushless motor without Hall sensor	
1257	The operation failed because the brushless motor is not commutated	Refer to the chapter on the **Adjuster Wizard** in the *SPiiPlus MMI Application Studio User Guide* for details on commutation.
1258	The operation failed because the motor is in open-loop mode	
1259	Motion cannot start because the motor is defined as dummy	
1260	Motion cannot start because the motor is disabled	The **ENABLE/ENABLEALL** command has to be run.
1261	Motion cannot start because the brushless motor is not commutated	Refer to the chapter on the **Adjuster Wizard** in the *SPiiPlus MMI Application Studio User Guide* for details on commutation.
1262	Motion cannot start because the motor is in open-loop mode	
1263	Motion cannot start because the previous motion failed. Use FCLEAR command.	To start the motion enter the **FCLEAR** command.
1265	SP program does not support this operation	
1266	Invalid PEG pulse width	Pulse width set in **PEG_I** or **PEG_R** is incorrect.
1267	Maximal number of time-based PEG pulses is 127	Not relevant to SPiiPlus NT PEG operations.

Table 63 ACSPL+ Syntax Errors (page 17 of 17)

Error Code	Error Message	Remarks
1268	Maximal period of time-based PEG pulses is 0.012 millisecond	Not relevant to SPiiPlus NT PEG operations.
1269	PEG pulse width must be less than time-based pulse period	Not relevant to SPiiPlus NT PEG operations.
1270	PEG is not supported for the specified axis	See **ASSIGNPEG**.
1271	Step does not agree with start/end points	
1272	Incremental PEG step must be specified in encoder counts within the (-2^31, 2^31-1) range, zero is excluded.	The PEG step in encoder counts must be in the range of -2^{31} to $2^{31}-1$.
1273	The specified axis does not exist in this controller model	Refers to controllers that support 4 axes only, and the specified axis is above this.
1274	The specified axis is defined as dummy only	
1275	Only stepper motor is supported for the specified axis	
1276	The brushless motor is not supported for the specified axis	
1277	Dual loop control is not supported for the specified axis	
1278	Remote HSSI driver is not supported for the specified axis	
1279	PEG states are not supported for the specified axis	
1280	The stepper motor is not supported for the specified axis	
1281	No Hall support for 2-Phase motors	
1282	Value out of range	
1283	Random PEG is limited to 256 points per axis at a time	See **PEG_R**.
1284	Axis is not connected to PEG engine	See **ASSIGNPEG**.

6.2 ACSPL+ Compilation Errors

The error codes in ACSPL+ compliation errors range between 2000 and 2999.

ACSPL+ compilation errors are reported either immediately when the erroneous line is inserted, or subsequently, when the controller attempts to compile an ACSPL+ program. If a program in a buffer undergoes compilation and an error is detected, the error code is stored in the corresponding element of the **PERL** array and the erroneous line number is stored in the corresponding element of the **PERL** array.

Table 64 ACSPL+ Compilation Errors (page 1 of 9)

Error Code	Error Message	Remarks
2002	Unrecognized command	The program line contains a sequence that is not recognized as a legal command. If the line starts from a standard variable name, check the following: ❑ All letters are uppercase ❑ Spelling is correct If the line starts from a user variable name, check the following: ❑ The variable is defined before use ❑ The letter case matches the definition ❑ Spelling is correct
2003	Unexpected delimiter, incomplete command	A command contained in the line is incomplete.
2006	Unexpected END	The end command is specified without a paired if, loop or while command.
2007	Two adjacent operators	Two binary operator are specified in the expression without an operand between.
2008	Left bracket was expected	Left bracket was not found in a position where it is expected. Possible reasons for this error are array name without index or function call without arguments.
2009	Right bracket was expected	The expression or sub-expression has an unpaired left bracket.
2010	Comma was expected	Comma character was not found in a position where it is expected. Possible reasons for this error are: ❑ Missing comma between the arguments of the command or function ❑ The command or function call has less arguments than required.

Table 64 ACSPL+ Compilation Errors (page 2 of 9)

Error Code	Error Message	Remarks
2011	Equals sign was expected	Equal sign in assignment is missing.
2012	Direct command was expected	
2013	DO or LOOP without END	
2015	Integer positive constant is expected	The declaration of array contains a size definition other than positive integer constant.
2016	Only label or line number is allowed as start point	The start command can contain the start line definition. The start line definition can appear only as a label or a positive line number.
2018	Scalar variable cannot be indexed or used with axis specification	The command contains a scalar variable followed by index specification or prefixed by axis specification.
2019	Write-only variable cannot be read	
2020	Read-only variable cannot be assigned	The command attempts to assign a read-only variable.
2021	Label was expected	**GOTO** or **CALL** must specify a label name.
2022	Array name was expected	The command or function requires an array name as one of its arguments. The array name must not be indexed.
2023	Variable name was expected	The declaration specifies an illegal variable name. A variable name must begin with a letter or underscore and can include letters, digits and underscores. A variable name must not include any of the following: ❑ Keywords (**IF**, **WAIT**, **SIN**, etc.) or standard variable names (**FPOS**, **MST**, etc.) ❑ Standard label (**AUTOEXEC**).
2024	Undefined label	**GOTO** or **CALL** specify a label name that was not defined in the program.
2025	Duplicated label	The program contains two identical labels.
2026	Undefined variable name	The command contains a variable name that was not declared in the program.

Table 64 ACSPL+ Compilation Errors (page 3 of 9)

Error Code	Error Message	Remarks
2027	Duplicated variable name	The program declares two variables with identical names.
2030	Number user array elements can not exceed the XARRSIZE parameter	The value of the **XARRSIZE** parameter is limited to a maximum of 30,000 elements. The limitation applies also to two-dimensional arrays. The total number of elements in a two-dimensional array is equal to the product of its sizes by each dimension. The total number of elements must not exceed 30,000.
2031	Global of different type/dimension was defined in other buffer	The program declares a global variable with the name that was already declared in other buffer with different type or dimension.
2033	Mandatory argument is omitted	The command or function call contains fewer arguments than required.
2034	More arguments than required	The command or function call contains more arguments than required.
2035	Wrong argument type	The command or function call contains an argument of incorrect type.
2036	Function that not returns value cannot be used in expression	The expression cannot include a function that has no return value.
2037	Axis specification was expected	This error refers to a logical check. The command must specify an axis. Axis specification can appear: as an integer designation, like 0, 1, 32, or as an expression that evaluates to an integer within the range of 0 and the number of axes in the system minus 1.
2038	Axis specification was expected	This error refers to a physical check. The command must specify an axis that physically exists. Axis specification can appear: as an integer designation, like 0, 1, 32, or as an expression that evaluates to an integer within the range of 0 and the number of axes in the system minus 1.
2043	Internal error	
2044	Index is out of range	The constant index value is greater than the array size minus one, or is negative.

Table 64 ACSPL+ Compilation Errors (page 4 of 9)

Error Code	Error Message	Remarks
2045	Illegal axis value	The axis value specified by a numerical constant is greater than the number of axes in system minus one, or is negative.
2048	Argument must be specified as + or - sign	The command requires a direction specification that must be presented by character + or -.
2049	Illegal suffix for this command	The command specifies a command option, which is illegal for this command.
2050	Name of standard variable was expected	The command requires an argument that specifies one of the ACSPL+ variables.
2051	Only APOS, RPOS, FPOS and F2POS are allowed in SET command	The **SET** command must specify assignment to one element of one of the following variables: **APOS**, **RPOS**, **FPOS**, **F2POS**. The variables must be in upper case.
2052	Variable name was expected	The command requires an argument that specifies a scalar variable or an element of array. A general expression or a constant is not allowed for this argument.
2053	Constant argument was expected	The command requires an argument that specifies a numerical constant.
2054	Illegal buffer number	The command must specify a constant buffer number. The constant must fall in the range of from 0 to 15.
2055	Assigned value is out of range	The command attempts to assign a variable with a value that falls out of the range allowed for the variable.
2056	Zero divide	The command attempts to divide by zero.
2057	Only VEL, ACC, DEC, JERK, KDEC are allowed in IMM command	**IMM** must specify assignment to one element of one of the following variables: **VEL**, **ACC**, **DEC**, **JERK**, **KDEC**.
2058	Bit selection cannot be applied to real variable	The bit selector is specified for a real variable.
2059	ELSE without IF	**ELSE** is specified without corresponding **IF** (see **IF, ELSEIF, ELSE...END**).

Table 64 ACSPL+ Compilation Errors (page 5 of 9)

Error Code	Error Message	Remarks
2060	ELSEIF without IF	**ELSEIF** is specified without corresponding **IF** (see **IF, ELSEIF, ELSE...END**).
2061	LOOP without END	**LOOP** has no corresponding **END** (see **LOOP...END**).
2063	IF without END	**IF** has no corresponding **END** (see **IF, ELSEIF, ELSE...END**).
2064	Memory overflow	The application requires too much memory. Reduce the number and size of the local and global variables used in the application, or use a more powerful model of the controller.
2065	Axis constant or axis expression in brackets expected	The command must include an axis specification in parentheses, for example: **FMASK(4).16**, where 4 is the axis specification.
2066	Too many axis specifiers	The axis specification includes too many axis specifiers.
2067	An axis is specified more than once	The axis specification includes two identical axis specifiers.
2068	Sign constant or sign expression in brackets expected	The command must include a sign specification. Each sign in the specification defines a direction of one involved axis. Sign specification can appear: ❑ As a sequence of sign characters, like +, +-+, -+-+ ❑ As a list of expressions enclosed in brackets where (0, 1, 0) is equivalent to +-+.
2069	Too many sign specifiers	The sign specification includes too many sign specifiers.
2070	Unknown #-constant	The command includes an unknown symbolic constant.
2071	Local variable is not allowed in this command	The command does not allow the use of local variables. Only ACSPL+ variables and user global variables that have been declared can be used.
2072	WHILE without END	**WHILE** has no corresponding **END** (see **WHILE...END**).

Table 64 ACSPL+ Compilation Errors (page 6 of 9)

Error Code	Error Message	Remarks
2073	WAIT, TILL, GOTO, CALL, LOOP..END, WHILE..END, INPUT are not allowed in immediate command	The specified commands cannot be entered through the SPiiPlus MMI Application Studio **Communication Terminal**.
2074	Only RPOS is allowed after CONNECT	**CONNECT** must specify assignment to one element of one of the **RPOS** variables.
2075	Only MPOS variable is allowed after MASTER	**MASTER** must specify assignment to one element of one of the **MPOS** variables.
2076	Constant bit selector is greater than 31 or less than 0	The command includes a constant bit selector. The value of bit selector must fall into the range from 0 to 31.
2077	Array name requires indexing	The array name must be followed by an index specification. For a two-dimensional array, two index specifiers are required.
2078	Current empty space in the dynamic buffer is not enough for the command	The dynamic buffer contains too many commands queued for execution. The application must wait for the controller to execute and remove one or more commands from the buffer.
2079	GOTO,CALL,RET,LOOP, WHILE,IF,ON are not allowed in dynamic buffer	Commands **GOTO**, **CALL**, **LOOP...END**, **WAIT**, **IF, ELSEIF, ELSE...END**, **ON...RET** cannot be used in the dynamic buffer.
2080	Local variable declaration is not allowed in a immediate command.	Local variables cannot be used in a SPiiPlus MMI Application Studio **Communication Terminal** command.
2081	Variable declaration is not allowed in dynamic buffer	Variable declaration cannot be executed in dynamic buffer.
2082	Illegal file name	The command requires file name specification.
2083	Integer overflow	The result of constant integer expression is more than 2147483647 or less than -2147483648. Consider using real constants instead of integer. To change constant type to real, add decimal point to the end of the constant.

Table 64 ACSPL+ Compilation Errors (page 7 of 9)

Error Code	Error Message	Remarks
2084	Integer constants are allowed in the range from -2147483648 to +2147483647	Consider using real constants instead of integer. To change constant type to real, add decimal point to the end of the constant.
2085	Protection violation	The controller is in the Protected mode and the command violates one of the protection rules (see **PROTECT**).
2086	Protection attribute cannot be changed for this parameter	**SETPROTECTION** cannot be executed for this ACSPL+ variable.
2087	Only constant 0 or 1 is allowed at the right side	The command allows only 0 or 1 to the right of the equals sign.
2088	No dual-port RAM in this controller	SPiiPlus NT does not contain a dual-port RAM. If the program has been imported from an older non-NT version and this error appears, delete any DPR command (such as **COPYFROMDPR** or **COPYTODPR**).
2089	Bit selection is not available for DPR variable	SPiiPlus NT does not contain a dual-port RAM. If the program has been imported from an older non-NT version and this error appears, delete any DPR command (such as **COPYFROMDPR** or **COPYTODPR**).
2090	Only global variables can be defined in DPR	SPiiPlus NT does not contain a dual-port RAM. If the program has been imported from an older non-NT version and this error appears, delete any DPR command (such as **COPYFROMDPR** or **COPYTODPR**).
2091	DPR address must be specified	SPiiPlus NT does not contain a dual-port RAM. If the program has been imported from an older non-NT version and this error appears, delete any DPR command (such as **COPYFROMDPR** or **COPYTODPR**).
2092	Only even numbers from 128 to 504 are allowed as DPR address	SPiiPlus NT does not contain a dual-port RAM. If the program has been imported from an older non-NT version and this error appears, delete any DPR command (such as **COPYFROMDPR** or **COPYTODPR**).
2093	Collision with other variable in DPR	SPiiPlus NT does not contain a dual-port RAM. If the program has been imported from an older non-NT version and this error appears, delete any DPR command (such as **COPYFROMDPR** or **COPYTODPR**).

Table 64 ACSPL+ Compilation Errors (page 8 of 9)

Error Code	Error Message	Remarks
2094	DPR variable is not allowed in this command	SPiiPlus NT does not contain a dual-port RAM. If the program has been imported from an older non-NT version and this error appears, delete any DPR command (such as **COPYFROMDPR** or **COPYTODPR**).
2095	Illegal line number	The specified line number must be a positive non-zero integer.
2096	Only even numbers from 128 to 1016 are allowed as DPR address	SPiiPlus NT does not contain a dual-port RAM. If the program has been imported from an older non-NT version and this error appears, delete any DPR command (such as **COPYFROMDPR** or **COPYTODPR**).
2097	Illegal SP number	Valid SP numbers are 0, 1, 2, and 3.
2098	Illegal SP variable specification	Invalid SP variable was entered.
2099	Undefined SP variable	Non-SP variable was entered.
2100	User-defined array was expected	
2101	Illegal character constant	
2102	Illegal tag	
2103	Tag can be specified only for global variables	
2104	BLOCK cannot be nested	See **BLOCK...END**.
2105	BLOCK without END	See **BLOCK...END**.
2106	Illegal declaration	
2107	Axis redefinition	
2108	Assignment to constant	Value has to be assigned to a variable.
2109	ALL cannot be used in this command	
2110	Ambiguous axis specification	
2111	Illegal format of real constant	
2112	This function cannot be used as argument of CONNECT, TRIGGER or in autoroutine condition	See **CONNECT** or **TRIGGER**

Table 64 ACSPL+ Compilation Errors (page 9 of 9)

Error Code	Error Message	Remarks
2113	Shared memory variables must be global	
2114	Shared memory variables can be declared only in the D-Buffer	

6.3 ACSPL+ Runtime Errors

The ACSPL+ runtime error code values range between 3000 and 3999.

Note	
	Codes from 3000 to 3019, however, do not indicate an error. For example, code 3002 reports that the user has terminated the program. *Codes from 3020 to 3999 indicate run-time errors.*

If an error occurs in immediate execution of ACSPL+ command, the error is indicated immediately in the prompt. If an error occurs when an ACSPL+ program is executed in a buffer, no immediate indication is provided. Instead, the error code and the line number are stored in the corresponding elements of the **PERL** and **PERL** arrays.

Table 65 ACSPL+ Runtime Errors (page 1 of 12)

Error Code	Error Message	Remarks
3023	Read-only variable cannot be assigned	A command specifies an assignment to a read-only variable.
3028	Scalar variable cannot accept axis specification	A scalar variable cannot be prefixed with an axis specification.
3034	Illegal index value	The command specifies an assignment to a read-only variable.
3035	Index is out of range	The reason of the error is one the following: ❑ The specified index value is more or equal to the number of elements in the array ❑ The specified index value is negative ❑ The specified index values are incompatible (first value in the range greater than last).

Table 65 ACSPL+ Runtime Errors (page 2 of 12)

Error Code	Error Message	Remarks
3037	Illegal variable name	The command requires specification of a standard variable name, but the specified name is not a name of an ACSPL+ variable.
3040	Unable to open file	The command specifies a file in the flash memory that does not exist.
3041	Assigned value is out of range	The command attempts to assign the standard variable with a value that is out of the range allowed for this variable.
3043	Program cannot start because the buffer was not compiled	The command attempts to start an ACSPL+ program that has not been compiled. To compile a program, in the SPiiPlus MMI Application Studio: ❑ In the **Program Manager**, right-click the buffer and select **Compile Buffer**, or ❑ Use the **#nC** command in the **Communication Terminal**, where **n** is the buffer number.
3044	Command cannot be executed while the program is running	The command attempts to affect a running ACSPL+ program. Stop the program before executing the command. To stop a program, in the SPiiPlus MMI Application Studio: ❑ In the **Program Manager**, right-click the buffer and select **Stop Buffer**, or ❑ Use the **#nS** command in the **Communication Terminal**, where **n** is the buffer number.
3045	Numerical error in standard function	The command includes an ACSPL+ function that caused a numerical error. Check if the arguments of the function fall into the allowed range.
3046	Write file error	The command caused a failed write to flash memory. A re-occurring error of this type indicates a serious problem in the controller hardware or firmware.
3047	Read file error	The command caused a failed read from flash memory. A re-occurring error of this type points to a serious problem in the controller hardware or firmware.

Table 65 ACSPL+ Runtime Errors (page 3 of 12)

Error Code	Error Message	Remarks
3048	More axes than were defined in the motion	The **POINT**command specifies more axes than were specified in the motion that the command refers to.
3050	Conflict with user-defined axis group	The command is incompatible with a previously defined user-defined axis group. The axes specified in the command may either belong to one user-defined axis group, or may not intersect with a user-defined axis group.
3051	Line number is out of range	The command specifies a line number that does not exist in the specified program buffer.
3052	Buffer number is out of range	The command specifies an illegal buffer number. The controller has program buffers numbered 0 to 15, where 15 is the D Buffer..
3053	Wrong type	The command addresses a standard variable of a different type from the variable specified in the command. This error never occurs when the user communicates with the controller through a communication terminal. The error occurs only when an application communicates with the controller using the SPiiPlus C Library. The error indicates a communication problem.
3055	Command requires line number specification	The command must contain a line number specification.
3060	Illegal memory command	The memory management command is improperly formatted.
3061	')' wasn't found	The command contains a non-paired left bracket.
3063	Variable is not defined in the buffer	The command addresses a variable that is not declared in the specified buffer, or the specified buffer is not compiled.
3064	Undefined global variable	The command addresses a global variable that is not declared in any buffer, or the buffer that contains the declaration is not compiled.

Table 65 ACSPL+ Runtime Errors (page 4 of 12)

Error Code	Error Message	Remarks
3065	Command cannot be executed while the current motion is in progress	The command is in conflict with one or more of the currently executed motions. To kill a motion use **KILL**.
3067	GO command failed	The controller has no motions waiting for **GO**.
3068	Referenced axis does not execute a motion (motion was terminated?)	The command specifies an axis, but no motion was specified for this axis.
3069	This command can be used only with MPTP, PATH, or PVSPLINE motion	The command specifies an axis, but the motion specified for the axis is incompatible with the command.
3070	Attempt to add segment after **ENDS** command	The command attempts to add a segment or a point to the motion that has already been closed by **ENDS**.
3071	File name was expected	The command must specify the name of an internal file in the flash memory.
3072	Wrong array size	The command specifies an array, but the motion that the command refers to, or other command arguments require an array of another size.
3073	Text for search is not specified	The command must specify a text for search operation.
3074	Only standard or SP variable is allowed in the command	The command requires an ACSPL+ or SP variable name to be specified.
3076	Undefined label	The command requires a label specification. The program that contains the label specified in the command must be compiled in a buffer.
3077	Protection violation	The command attempts to assign a protected variable when the controller is in protected mode. The controller must be in configuration mode before protected variables can be assigned (see **PROTECT**).

Table 65 ACSPL+ Runtime Errors (page 5 of 12)

Error Code	Error Message	Remarks
3078	Variable can be changed only while the motor is disabled	The command attempts to assign a variable that can be changed only if the motor is disabled. ❑ If the variable is axis-related, disable the corresponding motor before assigning the variable. ❑ If the variable is not axis-related, disable all motors before assigning the variable.
3079	Motion cannot start because one or more motors are disabled	The motion command involves one or more motors that are disabled.
3080	Default Connection flag is set for the motor	The command cannot be executed because the default connection flag is set for the motor. The default connection flag is bit 17, i.e., **#DEFCON**, of the **MFLAGS** variable (see **MFLAGS**).
3081	Incompatible suffixes	The command includes command options that cannot be used together.
3085	Extra number after the command	The command specifies a superfluous numerical argument.
3086	Variable name must be specified	The command requires a variable name specification.
3087	Command cannot be executed while the axis is moving	The command specifies one or more axes that are involved in a motion. The command can be executed only after the motion finishes.
3088	Variable can be queried only in compiled buffer	The command attempts to query a variable in a buffer that was not compiled.
3089	Label can be referenced only in compiled buffer	The command attempts to reference a label in a buffer that was not compiled.
3090	This type of motion is not allowed for grouped axis	The slave or track command specifies an axis that is included in a user-defined group. Only a single axis can be specified by **SLAVE** or **TRACK**.
3091	Less arguments than required	The motion command specifies less coordinate values than required by the axis specification.

Table 65 ACSPL+ Runtime Errors (page 6 of 12)

Error Code	Error Message	Remarks
3092	More arguments than required	The motion command specifies more coordinate values than required by the axis specification.
3093	Bit selector value is greater than 31 or less than 0	The expression specified in the bit selector yields a value greater than 31 or less than 0.
3096	'}' was not found	The command includes a non-paired left curly bracket.
3097	Previous data collection is in progress	The command attempts to start a data collection while the previous data collection for the same axis is still in progress.
3098	Stalled motion requires limits specification	The motion command, which includes a command option of stalled motion, requires specification of the motion limits.
3099	Extra numbers after the command	The motion command includes superfluous numerical arguments.
3101	The program is suspended, the start line number is not allowed	The command attempts to start a program from a specified line when the program is in suspended state. A program in suspended state can be only resumed from the next line. The line specification is not allowed. The only way to start a suspended program from a specific line is to stop the program, and to start it again from the desired line.
3105	Format error	The command is incorrectly formatted.
3106	SP function failed	The function that accesses SP memory cannot be executed. Check the address specified in the function call. The address must fall in the range of 0 to 511.
3108	Invalid real number	The command includes specification of a real number that cannot be interpreted as a valid real number.
3112	No matches were found	The search command did not find any matches.
3114	The program finished without a STOP command	The program that is running in a buffer executed the last command, and it was not a **STOP** command.

Table 65 ACSPL+ Runtime Errors (page 7 of 12)

Error Code	Error Message	Remarks
3115	Stack underflow (RET without CALL)	The program that runs in a buffer caused a stack underflow. Program execution requires **CALL**.
3116	Stack overflow (too many nested CALLs)	The program that is running in a buffer caused stack overflow. This occurs if the program executes two many nested calls. Check that the program does not specify infinite recursion (subroutine is in an infinite loop).
3117	Attempt to enter autoroutine directly	The program that is running in a buffer comes to the **ON** command (see **ON...RET**). **ON** must never be executed in the normal program flow.
3118	Illegal axis number	The command specifies an axis by number or expression, and the resulting value is not a legal axis number. Valid axis numbers range between 0 to the number of axes in the system minus one.
3120	The motion must involve the first two axes from the group	The segmented motion must always involve two axes. If a user-defined group contains more than two axes, the segmented motion must involve the first two axes in the group.
3121	Unknown #-constant	The command specifies an unknown symbolic constant.
3122	Bit selection is not applicable	Bit selector cannot be applied to a real value.
3123	Illegal bit selector	Value of bit selector must fall into the range 0..32.
3124	Attempt to enable motor failed	**ENABLE/ENABLEALL** failed. Additional information can be obtained from the corresponding element of the **MERR** variable (motor disable code).
3125	Error in SP program	The command caused unsuccessful loading of an SP program. The file with the SP program contains an error.
3126	Illegal SP number	The command specifies an illegal SP number. Legal SP numbers are from 0 to 3.

Table 65 ACSPL+ Runtime Errors (page 8 of 12)

Error Code	Error Message	Remarks
3127	Editing of this buffer is disabled	The command attempts to open the buffer for editing or to change the program in the buffer. Editing of a buffer is disabled if bit 1 of the corresponding element of **PFLAGS** = 1.
3129	In binary command the name specification must end with /	The command syntax requires a name to be followed by the / (slash) character.
3130	Segment sequence for the previous motion was not terminated with ENDS	**MPTP**, **MSEG**, **PATH** are followed by a number of the point or segment commands - segment definition sequence. The segment definition sequence must be closed with ENDS. See **MPTP...ENDS**, **MSEG...ENDS**, and **PATH...ENDS**.
3132	The file is not a legal user data file	**READ** specifies a file in the flash memory that is not a legal user data file. Only files created with **WRITE** are legal user data files.
3133	Discrepancy in types of the saved and restored arrays	The array and the user data file specified by **READ** contain data of different types. **READ** must specify an array of the same type and dimension as the array that was specified by the **WRITE** command that created the file. For example, the error occurs if a real array was saved in the file by **WRITE**, but **READ** tries to load the file into an integer array.
3134	Discrepancy in sizes of the saved and restored arrays	The array and the user data file specified by **READ** are different sizes. **READ** must specify an array of the same type and dimension as the array that was specified by the **WRITE** command that created the file.
3136	Wrong relation between first point, last point and interval	The interval value specified by **PEG_I** or **PEG_R** does not correspond to the start and final points. For example, the error occurs if the final point is less than the start point, but the step is positive.
3137	Illegal analog output number	The command specifies a number of analog outputs which is not available in the controller.

Table 65 ACSPL+ Runtime Errors (page 9 of 12)

Error Code	Error Message	Remarks
3138	Incompatible SP and analog output	The command specifies an analog output number that cannot be accessed in the specified SP.
3139	Illegal input	The controller tries to interpret the string as a response to the executed input command but the string does not follow the required format.
3142	Arguments are inconsistent	Check arguments against the specifications for the **PEG_I** or **PEG_R** commands.
3144	Simulator does not support this command	The command cannot be executed by the simulator.
3145	The specified DPR address is less than 128 or exceeds the DPR size	SPiiPlus NT does not contain a dual-port RAM. If the program has been imported from an older non-NT version and this error appears, delete any DPR command (such as **COPYFROMDPR** or **COPYTODPR**).
3146	Collision with other variable in DPR	SPiiPlus NT does not contain a dual-port RAM. If the program has been imported from an older non-NT version and this error appears, delete any DPR command (such as **COPYFROMDPR** or **COPYTODPR**).
3149	Illegal SP address	The SP address must fall in the range of from 0 to 512.
3150	Only even numbers are allowed as DPR address	SPiiPlus NT does not contain a dual-port RAM. If the program has been imported from an older non-NT version and this error appears, delete any DPR command (such as **COPYFROMDPR** or **COPYTODPR**).
3151	This is a DEMO version of Simulator. 5 minutes of work remains.	The demo version of the simulator has a limited session time. The simulator is going to stop after five minutes.
3152	The DEMO version of Simulator has terminated	The demo version of the Simulator has a limited session time. The session time has elapsed.
3154	This command can be used only with MSEG motion	Only **PROJECTION**, **LINE**, **ARC1**, **ARC2**, and **STOPPER** commands apply to **MSEG...ENDS**.

Table 65 ACSPL+ Runtime Errors (page 10 of 12)

Error Code	Error Message	Remarks
3155	Motion cannot start because the previous motion failed. Use FCLEAR command.	In the Strict mode (**S_FLAGS.#FCLEAR** = 1) the motion cannot start after a fault has occurred. Use the command **FCLEAR** to clear the result of the fault.
3201	End-of-Sequence is illegal for this motion	The ends command cannot be specified for this type of motion.
3212	ARC arguments are inconsistent	**ARC1** or **ARC2** specify inconsistent arguments. The desired arc cannot be calculated.
3213	Stopper is prohibited for master-slave motion	**STOPPER** cannot be used with segmented **MASTER-SLAVE** motion.
3214	Adjacent stoppers are prohibited	Two adjacent **STOPPER** commands are not allowed.
3215	In cyclic path the first and the last points must coincide	In a cyclic segmented motion, the first point of the first segment must coincide with the last point of the last segment.
3216	Velocity is specified, but the motion command had no V command option	Velocity argument can be specified in **POINT**, **LINE**, **ARC1** or **ARC2** only if the corresponding motion command specifies **/v**.
3217	Segment of zero length	Segments of zero length are not allowed.
3231	Illegal key	The **Key** argument of the **GETCONF** or **SETCONF** command specifies a value that is not supported in this controller.
3232	Illegal index	The **Index** argument of the **GETCONF** or **SETCONF** command specifies a value that is not supported in this controller.
3233	Illegal value	The **Value** argument of the **SETCONF** command specifies a value that is not compatible with the **Key** argument.
3234	Value in SETCONF must be 0 or 1	For the specified **Key** only 0 or 1 are legal in the **Value** argument. of **SETCONF**
3235	SETCONF function is disabled	The specified **SETCONF** function is disabled in the current controller mode.
3236	SETCONF cannot be executed while the motor is enabled	The specified **SETCONF** function cannot be executed while the motor is enabled.

Table 65 ACSPL+ Runtime Errors (page 11 of 12)

Error Code	Error Message	Remarks
3253	Unable to work with Dummy motor	The command cannot be executed for a dummy motor. The addressed motor is configured as a dummy, (see the **#DUMMY** bit of **MFLAGS**).
3254	The operation requires the motor to be enabled	The command can be executed only while the motor is enabled.
3255	The operation requires the motor to be disabled	The command can be executed only while the motor is disabled.
3256	The operation is valid only for brushless motor without Hall sensor	The command cannot be executed for any motor type other than brushless, and only for brushless motors that do not have a Hall sensor.
3257	The operation failed because the brushless motor is not commutated	The command cannot be executed because the defined brushless motor is not commutated. The **#BRUSHOK** bit of **MFLAGS**. reflects the state of the brushless commutation. To commutate the motor, execute **COMMUT** or a specific initialization program.
3258	The operation failed because the motor is in open-loop mode	The command cannot be executed because the motor is in open-loop state. The open-loop mode may be set using the **#OPEN** bit of **MFLAGS**.
3259	Motion cannot start because the motor is defined as dummy	The motion command cannot be executed because one or more of the motors involved are dummy. Motor dummy state is defined by the **#DUMMY** bit of **MFLAGS**..
3260	Motion cannot start because the motor is disabled	The motion command cannot be executed because one or more of the motors involved are disabled.
3261	Motion cannot start because the brushless motor is not commutated	The command cannot be executed because the defined brushless motor is not commutated. The **#BRUSHOK** bit of **MFLAGS**. reflects the state of the brushless commutation. To commutate the motor, execute **COMMUT** or a specific initialization program.

Table 65 ACSPL+ Runtime Errors (page 12 of 12)

Error Code	Error Message	Remarks
3262	Motion cannot start because the motor is in open-loop mode	The motion command cannot be executed because one or more of the motors involved are in open-loop state. Open-loop state is defined by bit **MST.#OPEN**. The open-loop mode may be set using the **#OPEN** bit of **MFLAGS**.
3263	Motion cannot start because the previous motion failed. Use FCLEAR command.	The motion command cannot execute because the previous motion for one or more motors failed and the controller is set to strict mode. The Strict mode is defined by bit **S_FLAGS.#FCLEAR**. Use **FCLEAR** to clear the result of the fault.
3265	SP program does not support this operation	The command is not supported by the current version of the controller. Specifically, the SP program does not support the operation. Check if the SP program has been changed.

6.4 Motion Termination Errors

Motion Termination error code values range between 5000 and 5150.

Note	
	Codes from 5000 to 5008, however, do not indicate an error. They report normal motion termination. *Codes from 5009 and higher appear when a motion is terminated or a motor is disabled due to a fault detected by the controller.*

When a motion terminates abnormally, or a motor is disabled , the error code is stored in the **MERR** variable. If there is an initialization fault during startup, the error code is stored in the **S_ERR** variable.

Table 66 ACSPL+ Motion Termination Errors (page 1 of 5)

Error Code	Error Message	Remarks
5000	Motion has not finished	Motion was terminated before reaching final point.
5001	Motion generation finished	The motion came to the final point and was successfully completed.
5003	Motion was terminated by user	The motion was terminated by **HALT** before the final point was reached
5004	Motor was disabled by user	The motion was disabled by **DISABLE**.
5005	Motion was terminated because a motor was disabled	The motion was disabled by **DISABLE**.
5006	Motion was killed	The motion was terminated by **KILL** before the final point was reached
5007	Motor was disabled due to another motor becoming disabled	If a fault occurs in an axis disabling the motor thereby disabling other motors, code 5007 is stored in the **MERR** variable for all affected axes.
5008	Motion was killed due to another motion being killed	If a fault occurs in an axis killing the motion of a motor thereby killing the motion other motors, code 5007 is stored in the **MERR** variable for all affected axes.
5010	Right Limit	The motion was killed because of a right limit fault.

Table 66 ACSPL+ Motion Termination Errors (page 2 of 5)

Error Code	Error Message	Remarks
5011	Left Limit	The motion was killed because of a left limit fault.
5014	Overheat	The motor was disabled or the motion failed because of an overheat fault.
5015	Software Right Limit	The motion is killed because of a software right limit fault.
5016	Software Left Limit	The motion is killed because of a software left limit fault..
5017	Encoder Not Connected	The motor was disabled because of an encoder not connected fault.
5018	Encoder 2 Not Connected	The motor was disabled because of an encoder 2 not connected fault.
5019	Drive Alarm	The motor was disabled because of a drive alarm fault.
5020	Encoder Error	The motor was disabled or the motion failed because of an encoder error fault.
5021	Encoder 2 Error	The motor was disabled or the motion failed because of an encoder 2 error fault.
5022	Position Error	The motor was disabled or the motion failed because of a position error fault.
5023	Critical Position Error	The motor was disabled or the motion failed because of a critical position error fault.
5024	Velocity Limit	The motor was disabled or the motion failed because of a velocity limit fault.
5025	Acceleration Limit	The motor was disabled or the motion failed because of an acceleration limit fault.
5026	Overcurrent	The motor was disabled or the motion failed because of an overcurrent fault.
5027	Servo Processor Alarm	The motor was disabled or the motion failed because of a servo processor alarm fault.
5032	Attempt of motion while a fault is active	

Table 66 ACSPL+ Motion Termination Errors (page 3 of 5)

Error Code	Error Message	Remarks
5033	Attempt of Motion in Disabled Direction	The motion is killed because of an attempt to move to left direction when of Left Limit or Left Software Limit fault is On, or move to right direction when the Right Limit or Right Software Limit fault is On.
5035	Program Error	The motor was disabled or the motion failed because of a program error fault.
5036	Memory Overuse	The motor was disabled or the motion failed because of a memory overuse fault.
5037	Time Overuse	The motor was disabled or the motion failed because of a time overuse fault.
5038	Emergency Stop	The motor was disabled because of an emergency stop fault.
5039	Servo Interrupt	The motor was disabled or the motion failed because of a servo interrupt fault.
5060	Driver Alarm	Drive alarm: No fault
5061	Drive Alarm: Short Circuit	The motor was disabled or the motion failed because of a short circuit condition in the drive.
5062	Drive Alarm: External Protection Activated	The motor was disabled or the motion failed because of an external protection condition in the drive.
5063	Drive Alarm: Power Supply Too Low	The motor was disabled or the motion failed because of a power supply too low condition in the drive.
5064	Drive Alarm: Power Supply Too High	The motor was disabled or the motion failed because of a power supply too high condition in the drive.
5065	Drive Alarm: Temperature Too High	The motor was disabled or the motion failed because of a power supply too high condition in the drive.
5066	Drive Alarm: Power Supply 24VF1	The motor was disabled or the motion failed because of a power supply 24VF1 condition in the drive.
5067	Drive Alarm: Power Supply 24VF2	The motor was disabled or the motion failed because of a power supply 24VF2 condition in the drive.

Table 66 ACSPL+ Motion Termination Errors (page 4 of 5)

Error Code	Error Message	Remarks
5068	Drive Alarm: Emergency Stop	The motor was disabled or the motion failed because of an emergency stop condition in the drive.
5069	Drive alarm: Power-down	The motor was disabled or the motion failed because of a power-down condition in the drive.
5075	Drive Alarm: Digital Drive Interface not Connected	The motor was disabled or the motion failed because of an digital drive interface not connected condition in the drive.
5080	Component Failure	
5081	Component Failure: Unknown error	
5082	Component Failure: Unknown error	
5083	Component Failure: Unknown error	
5084	Component Failure: Power supply too high	
5085	Component Failure: Temperature too high	
5086	Component Failure: Unknown error	
5087	Component Failure: Unknown error	
5088	Component Failure: Unknown error	
5089	Component Failure: Power down	
5090	Component Failure: Drive supply phase lost	
5091	Component Failure: Power supply not ready. Try to enable axis again within 10 seconds	
5092	Component Failure: Unknown error	
5093	Component Failure: Unknown error	
5094	Component Failure: Power supply damper not OK	
5095	Component Failure: Unknown error	

Table 66 ACSPL+ Motion Termination Errors (page 5 of 5)

Error Code	Error Message	Remarks
5100	Current bias measured is out of range	At the beginning of the enable process the controller measures the current and calculates an average bias (offset). If the result is greater than the predefined maximum value (2% of maximum output), the controller generates the error and axis remains disabled.
5101	Autocommutation failure (phase error)	Error occurs if **COMMUT** fails. The error indicates wrong phase sequencing, or incorrect commutation parameters. To avoid this error it is recommended that the initial commutation be performed from the Adjuster. The Adjuster commutation program verifies the phase sequence and commutation parameters. This must be done before **COMMUT** is executed.
5102	Autocommutation failure (position error)	Error occurs if **COMMUT** fails. The error indicates bad or marginal servo tuning, or incorrect commutation parameters. To prevent this error make sure that the axis is properly tuned - use the **Adjuster Wizard** or **FRF Analyzer**.
5103	ENABLE is prohibited while Constant Current is ON	
5104	ENABLE failed because the motor is moving	**ENABLE** cannot be executed whenever the motor is in motion (see **ENABLE/ENABLEALL**).
5105	E_TYPE does not match HSSI-HES DIP switch settings	Check the HSSI-HES DIP switch settings.
5106	EtherCAT node failure	

6.5 System Errors

System Errors error code values range between 5151 and 5999.

Table 67 ACSPL+ System Errors (page 1 of 4)

Error Code	Error Message	Remarks
5151	Integrity of the firmware or user application is broken	
5152	Initialization problem: Main Interrupt is not detected	
5154	Initialization problem: Main Interrupt period is wrong	
5156	Initialization problem: Main Interrupt Event cannot be created	
5158	Initialization problem: SP initialization failed	
5160	Initialization problem: SP program activation failed	
5162	Initialization problem: SP and MP are not synchronous	
5164	Initialization problem: Improper controller card is detected	
5165	Initialization problem: Improper configuration file is detected	
5166	Initialization problem: SP is not detected by hardware	
5167	Initialization problem: Number of drive modules does not meet the configuration	
5168	Initialization problem: Drive Interface is not connected	
5169	Initialization problem: One or more parameters are out of range for the current controller configuration. Default values are assigned	

Table 67 ACSPL+ System Errors (page 2 of 4)

Error Code	Error Message	Remarks
5170	Initialization problem: One or more EEPROM parameters are different from the current controller configuration. Values from EEPROM are assigned	
5171	Initialization problem: One axis attached to two drives in configuration file	
5172	Initialization problem: Nominal current parameter in configuration file is different from the current controller configuration. Value from EEPROM is assigned	
5173	Initialization problem: Peak current parameter in configuration file is different from the current controller configuration. Value from EEPROM is assigned	
5174	Initialization problem: Power parameter in configuration file is different from the current controller configuration. Value from EEPROM is assigned	
5175	Initialization problem: Voltage (min or max) parameter in configuration file is different from the current controller configuration. Value from EEPROM is assigned	
5176	Initialization problem: Type parameter in configuration file is different from the current controller configuration. Value from EEPROM is assigned	
5177	Initialization problem: Number of subsystems parameter in configuration file is different from the current controller configuration. Value from EEPROM is assigned	
5178	Initialization problem: Component was detected by I2C but was not found in configuration file	

Table 67 ACSPL+ System Errors (page 3 of 4)

Error Code	Error Message	Remarks
5179	Initialization problem: Slave address parameter in configuration file is different from the current controller configuration. Value from EEPROM is assigned	
5180	Initialization problem: Drive number parameter in configuration file is different from the current controller configuration. Value from EEPROM is assigned	
5181	Initialization problem: RMS protection time constant parameter in configuration file is different from the current controller configuration. Value from EEPROM is assigned	
5183	Initialization problem: Number of axes parameter in configuration file is different from the current controller configuration. Value from EEPROM is assigned	
5184	Initialization problem: One of digital inputs parameters in configuration file is different from the current controller configuration. Value from EEPROM is assigned	
5185	Initialization problem: One of digital outputs parameters in configuration file is different from the current controller configuration. Value from EEPROM is assigned	
5186	Initialization problem: One of analog inputs parameters in configuration file is different from the current controller configuration. Value from EEPROM is assigned	
5187	Initialization problem: One of analog outputs parameters in configuration file is different from the current controller configuration. Value from EEPROM is assigned	

Table 67 ACSPL+ System Errors (page 4 of 4)

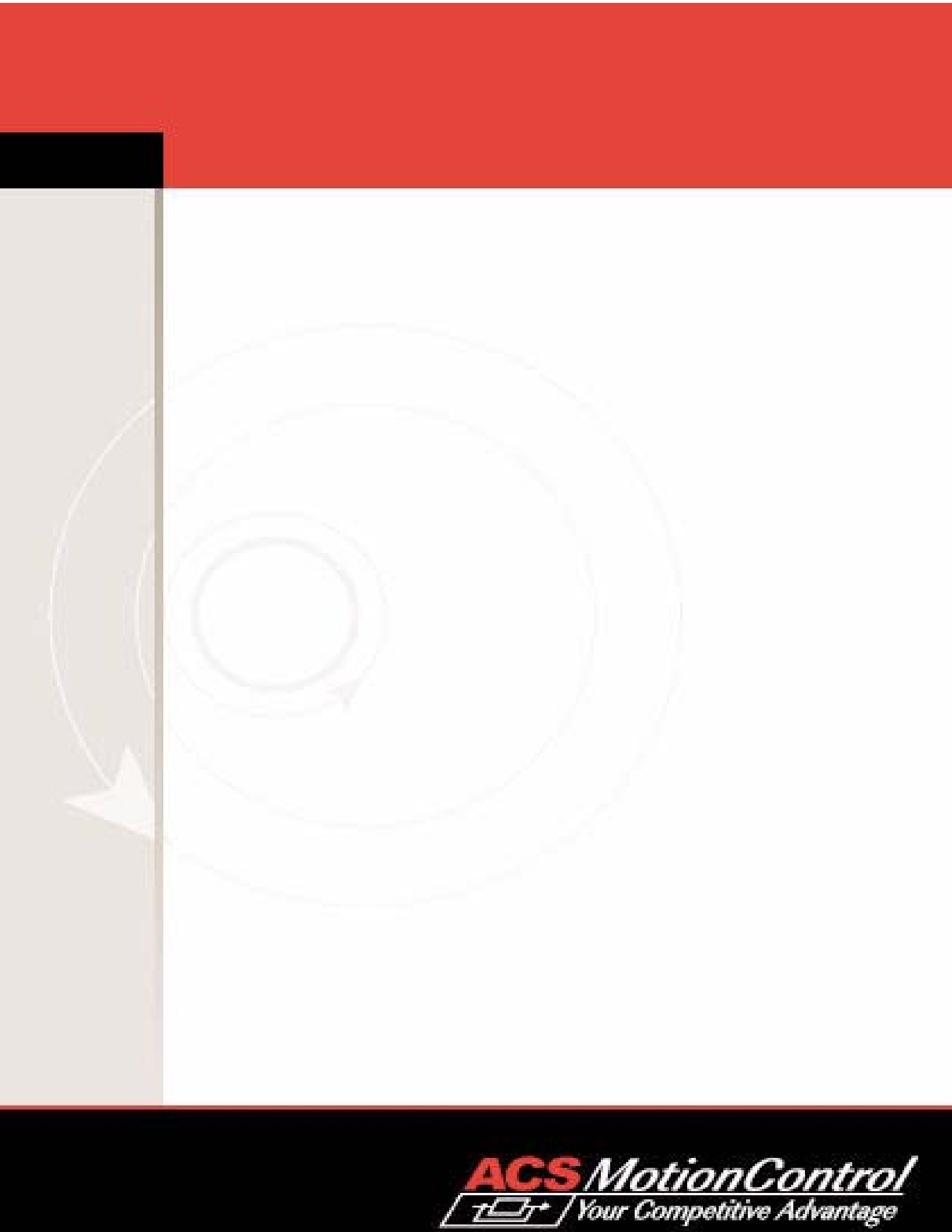
Error Code	Error Message	Remarks
5188	Initialization problem: Number of HSSI channels parameter in configuration file is different from the current controller configuration. Value from EEPROM is assigned	
5189	Initialization problem: Unit ID is wrong. The unit definition is ignored	
5190	Initialization problem: Number of nodes does not meet the configuration	
5191	Initialization problem: Number of axes does not meet the configuration	
5201	Initialization problem: Wrong range of input parameters. Servo Processor program activation failed	
5202	Initialization problem: File not found. Servo Processor program activation failed	
5203	Initialization problem: Read file error. Servo Processor program activation failed	
5204	Initialization problem: Memory allocation error. Servo Processor program activation failed	
5205	Initialization problem: Checksum error. Servo Processor program activation failed	
5206	Initialization problem: EtherCAT error. Servo Processor program activation failed	
5207	Initialization problem: EtherCAT timeout. Servo Processor program activation failed	

6.6 EtherCAT Errors

EtherCAT errors range from 6000 to 6999 and are latched in the **ECERR** variable.

Table 68 ACSPL+ EtherCAT Errors

Error Code	Error Message	Remarks
6000	General EtherCAT Error	
6001	EtherCAT cable not connected	
6002	EtherCAT master is in incorrect state	
6003	Not all EtherCAT frames can be processed	
6004	EtherCAT Slave error	
6005	EtherCAT initialization failure	
6006	EtherCAT cannot complete the operation	
6007	EtherCAT work count error	
6008	Not all EtherCAT slaves are operational	
6009	EtherCAT protocol timeout	
6010	Slave initialization failed	
6011	Bus configuration mismatch	
6012	CoE emrgency	
6013	EtherCAT Slave won't enter INIT state	


ACS MotionControl
Your Competitive Advantage