8SMC5-USB User Manual

Release 2.18.1

Standa

Mar 24, 2023

CONTENTS:

1 About **2**
- 1.1 General information . . . 2
- 1.2 Benefits . . . 3
  - 1.2.1 Main benefits . . . 3
  - 1.2.2 All benefits . . . 3
- 1.3 Table of specifications . . . 4
- 1.4 Specifications . . . 5
  - 1.4.1 Motor requirements . . . 5
  - 1.4.2 Electric specifications of the controller . . . 5
  - 1.4.3 Rotation control features . . . 6
  - 1.4.4 Additional firmware features . . . 6
  - 1.4.5 Additional features available via motor connector . . . 6
  - 1.4.6 Additional features available via backplane connector . . . 7
  - 1.4.7 Programming the controller . . . 7

2 Safety instructions **9**
- 2.1 Power supply and grounding requirements. Connection to controller . . . 9
- 2.2 Controller board . . . 11
- 2.3 One and two axes system in box . . . 11

3 Quick start guide **12**
- 3.1 Overview and getting started . . . 12
  - 3.1.1 Introduction . . . 13
  - 3.1.2 Requirements . . . 13
  - 3.1.3 Software installation and startup procedures . . . 17
  - 3.1.4 Getting started with XILab software . . . 18
  - 3.1.5 Functional test . . . 21
  - 3.1.6 Control from user applications . . . 21
- 3.2 Multiaxis system . . . 21
  - 3.2.1 Introduction . . . 22
  - 3.2.2 Required components . . . 23
  - 3.2.3 Getting started with multi-axis XiLab. . . . 24
  - 3.2.4 Multi-axis system temperature control. . . . 24
  - 3.2.5 Errata . . . 24
    - 3.2.5.1 Operation . . . 24
    - 3.2.5.2 Construction . . . 24
- 3.3 Example of a motor connection . . . 24
  - 3.3.1 General case . . . 25
  - 3.3.2 Example . . . 25
    - 3.3.2.1 Preparation . . . 25

3.3.2.2 Connecting the motor and encoder to the controller . . . 27
3.4 Manual profile setting . . . 31
3.4.1 Introduction . . . 31
3.4.2 Getting started . . . 31
3.4.3 Nominal current setting . . . 32
3.4.4 Basic parameters setting . . . 33
3.4.5 Hardware limit switches setting. Homing. . . . 33
3.4.6 Encoder parameters setting . . . 37
3.4.7 Setting the kinematic characteristics of the controller . . . 39
3.4.8 Working with user units . . . 39
3.5 Calculation of the nominal current . . . 40
3.5.1 Calculation based on the parameters of unipolar full step mode . . . 40
3.5.2 Calculation based on the parameters of bipolar full step mode . . . 40
3.5.3 The relationship with an rms current . . . 40
3.5.4 Amplitude and rated current for BLDC . . . 41
3.5.5 Setting the nominal current . . . 42

4 Technical specification **43**
4.1 Appearance and connectors . . . 43
4.1.1 Controller board . . . 43
4.1.1.1 Dimensions and arrangement . . . 43
4.1.1.2 Controller board connectors . . . 44
4.1.1.2.1 Stage connector . . . 44
4.1.1.2.2 Power supply connector . . . 46
4.1.1.2.3 Data connector . . . 46
4.1.1.3 Backplane connector . . . 47
4.1.2 One axis system . . . 49
4.1.2.1 Connectors . . . 49
4.1.2.1.1 Stage connector . . . 49
4.1.2.1.2 Power supply connector . . . 51
4.1.2.1.3 Data connector . . . 51
4.1.2.1.4 Supplementary one-axis system connector . . . 52
4.1.3 Two axes system . . . 54
4.1.3.1 Enclosure view . . . 54
4.1.3.2 Connectors . . . 54
4.1.3.2.1 Stage connector . . . 54
4.1.3.2.2 Power supply connector . . . 55
4.1.3.2.3 Data connector . . . 56
4.1.3.2.4 Joystick connector . . . 57
4.1.3.2.5 Supplementary two-axis system connector . . . 58
4.2 Kinematics and rotation modes . . . 59
4.2.1 Predefined speed rotation mode . . . 59
4.2.2 Rotation for predefined point . . . 60
4.2.3 Predefined displacement mode . . . 60
4.2.4 Acceleration mode . . . 60
4.2.5 Backlash compensation . . . 61
4.2.6 Rotation reversal . . . 62
4.2.7 Recommendations for accurate rotation . . . 62
4.2.8 PID-algorithm for DC engine control . . . 63
4.2.8.1 Algorithm description . . . 63
4.2.8.2 Particular properties of the algorithm . . . 63
4.2.8.2.1 PID regulator coefficients . . . 63
4.2.8.2.2 Reaching target position . . . 64
4.2.8.3 PID regulator tuning recommendations . . . 64

4.2.9 Feedback EMF . . . 65
4.2.9.1 Advantages . . . 65
4.2.9.2 Behavior of the engine when exposed to an external force . . . 66
4.2.9.3 Selecting L, R, and backEMF parameters for EMF algorithm . . . 66
4.2.9.4 The choice of PID coefficients for EMF . . . 68
4.2.9.4.1 Operation algorithm . . . 68
4.2.10 Feedback encoder . . . 68
4.2.11 Feedback encoder mediated . . . 69
4.2.12 Stop motion modes . . . 69
4.2.12.1 Immediate stop . . . 69
4.2.12.2 Stop with deceleration . . . 69
4.2.13 PID-algorithm for BLDC engine control . . . 69
4.2.13.1 PID-algorithm description . . . 69
4.2.13.2 PID regulator manual tuning . . . 70
4.2.13.2.1 Steps to adjust the coefficients: . . . 70
4.3 Main features . . . 71
4.3.1 Supported motor types . . . 71
4.3.1.1 Stepper motors . . . 71
4.3.1.2 DC motors . . . 72
4.3.1.3 BLDC motors . . . 72
4.3.1.4 Engine selection criteria . . . 73
4.3.2 Motor limiters . . . 75
4.3.3 Limit switches . . . 76
4.3.3.1 Limit switches designation . . . 76
4.3.3.2 General settings . . . 77
4.3.3.3 Programmable motion range limitation . . . 77
4.3.3.4 Hardware limit switches . . . 77
4.3.3.5 Limit switches connecting instructions . . . 77
4.3.3.6 Limit switches location on translators . . . 78
4.3.4 Automatic Home position calibration . . . 79
4.3.4.1 Standard homing algorithm . . . 79
4.3.4.2 Accurate additional calibration . . . 79
4.3.4.3 Fast homing algorithm . . . 80
4.3.4.4 Autocalibration features . . . 80
4.3.5 Operation with encoders . . . 80
4.3.5.1 Application of encoders . . . 80
4.3.5.2 What is quadrature encoder? . . . 81
4.3.5.3 Controller's features . . . 81
4.3.5.4 Encoder connection . . . 82
4.3.5.4.1 Operation with long cables . . . 83
4.3.5.4.2 Automatic encoder type detection . . . 83
4.3.6 Revolution sensor . . . 83
4.3.6.1 Connection diagram . . . 83
4.3.7 Steps loss detection . . . 84
4.3.8 Power control . . . 85
4.3.8.1 Current consumption reduction . . . 85
4.3.8.2 The motor power shutdown . . . 85
4.3.8.3 Time delay calculation specifics . . . 86
4.3.8.4 Jerk free function . . . 86
4.3.9 Critical parameters . . . 86
4.3.10 Saving the parameters in the controller flash memory . . . 87
4.3.11 User defined position units . . . 87
4.3.12 Usage of a coordinate correction table for more accurate positioning . . . 88
4.4 Safe operation . . . 88

4.4.1 Movement range bounds and limit switches . . . 89
4.4.2 Movement range limiters . . . 89
4.4.3 Critical Parameters . . . 89
4.4.4 Operation with Encoder . . . 89
4.5 Additional features . . . 90
4.5.1 Operating modes indication . . . 90
4.5.1.1 Controller status . . . 90
4.5.1.2 Indication of limit switches . . . 91
4.5.1.3 Connection diagram . . . 91
4.5.1.3.1 Controller board . . . 91
4.5.1.3.2 One-axis and two-axis systems . . . 92
4.5.2 Operations with magnetic brake . . . 93
4.5.2.1 Description of operation . . . 93
4.5.2.1.1 Controller operating sequence during stage shutdown. . . . 93
4.5.2.2 Magnetic brake connection diagram . . . 93
4.5.2.2.1 Controller board . . . 93
4.5.2.2.2 One-axis and two-axis systems . . . 94
4.5.3 Joystick control . . . 95
4.5.3.1 General information . . . 95
4.5.3.2 Connection diagram . . . 97
4.5.3.2.1 Controller board . . . 97
4.5.3.2.2 One-axis and two-axis systems . . . 97
4.5.4 Left-Right buttons control . . . 98
4.5.4.1 Connection diagram . . . 98
4.5.4.1.1 Controller board . . . 98
4.5.4.1.2 One-axis and two-axis systems . . . 99
4.5.5 TTL synchronization . . . 99
4.5.5.1 Principle of operation . . . 99
4.5.5.2 Connection . . . 101
4.5.5.3 Sync in . . . 101
4.5.5.4 Sync out . . . 104
4.5.5.5 Connection diagram . . . 108
4.5.5.5.1 Controller board . . . 108
4.5.5.5.2 One-axis and two-axis systems . . . 109
4.5.6 Multiaxis system design . . . 110
4.5.7 General purpose digital input-output (EXTIO) . . . 113
4.5.7.1 Connection diagram . . . 113
4.5.7.1.1 Controller board . . . 113
4.5.7.1.2 One-axis and two-axis systems . . . 114
4.5.8 General purpose analog input . . . 115
4.5.8.1 Connection diagram . . . 115
4.5.8.1.1 Controller board . . . 115
4.5.8.1.2 One-axis and two-axis systems . . . 115
4.5.9 External driver control interface . . . 116
4.5.9.1 Connection diagram . . . 117
4.5.9.1.1 Controller board . . . 117
4.5.9.1.2 One-axis and two-axis systems . . . 117
4.5.10 Serial port . . . 118
4.5.10.1 Serial port connection diagram . . . 119
4.5.10.1.1 Controller board serial port connection . . . 119
4.5.10.1.2 One-axis and two-axis systems serial port connection . . . 119
4.5.11 Saving the position in FRAM memory . . . 120
4.5.12 The Standa stages detection . . . 120
4.5.12.1 For developers . . . 121

4.5.12.2 Connection diagram for external memory test . . . 121
4.6 Secondary features . . . 121
4.6.1 Zero position adjustment . . . 121
4.6.2 User-defined position adjustment . . . 122
4.6.3 Controller status . . . 122
4.6.3.1 Movement status . . . 122
4.6.3.2 Motor power supply status . . . 122
4.6.3.3 Encoder status . . . 123
4.6.3.4 Motor windings status . . . 123
4.6.3.5 Position status. . . . 123
4.6.3.6 Controller power supply status and temperature. . . . 123
4.6.3.7 Status flags . . . 124
4.6.3.8 Digital signals status. . . . 125
4.6.4 USB connection autorecovery . . . 125
4.7 Software compatibility . . . 126
4.7.1 MicroManager . . . 126
4.7.1.1 Preparation . . . 126
4.7.1.2 Getting started with MicroManager . . . 127
4.7.1.2.1 Run MicroManager . . . 127
4.7.1.2.2 Configure hardware . . . 128
4.7.1.2.3 Device usage . . . 132
4.7.2 TANGO . . . 133
4.7.2.1 Overview . . . 133
4.7.2.2 Declaring the device, configuring and starting device server . . . 134

5 XILab application User's guide 139
5.1 About XILab . . . 139
5.2 Main windows of the XILab application . . . 139
5.2.1 XILab Start window . . . 139
5.2.2 XILab Main window in single-axis control mode . . . 143
5.2.2.1 Motion Control Unit . . . 146
5.2.2.1.1 Movement without specifying the final position . . . 146
5.2.2.1.2 Movement to the target point . . . 147
5.2.2.1.3 Target position for motion commands . . . 147
5.2.2.2 Attenuator Control Unit . . . 148
5.2.2.3 Controller and motor status . . . 149
5.2.2.3.1 Controller Power Supply . . . 149
5.2.2.3.2 Motor status . . . 149
5.2.2.3.3 Program status . . . 150
5.2.2.4 Group of application control buttons . . . 150
5.2.2.5 Status bar . . . 151
5.2.3 XILab Main window in multi-axis control mode . . . 152
5.2.3.1 Motion control block . . . 153
5.2.3.2 Virtual joystick block . . . 154
5.2.3.3 Control block . . . 155
5.2.3.4 Block of status indicators for controllers and motors . . . 156
5.2.4 Application settings . . . 157
5.2.5 Charts . . . 159
5.2.5.1 Values displayed on the charts . . . 161
5.2.5.2 Button functions . . . 161
5.2.5.3 Limit value . . . 162
5.2.6 Scripts . . . 162
5.2.6.1 Button functions . . . 163
5.2.7 XILab log . . . 164

5.3 Controller Settings . . . 165
5.3.1 Settings of kinematics (stepper motor) . . . 165
5.3.1.1 Motor parameters - directly related to the electric motor settings . . . 165
5.3.1.2 Motion setup - movement kinematics settings . . . 166
5.3.1.3 Feedback settings . . . 166
5.3.2 Motion range and limit switches . . . 166
5.3.3 Critical board ratings . . . 168
5.3.4 Power consumption settings . . . 169
5.3.5 Home position settings . . . 171
5.3.6 Synchronization settings . . . 172
5.3.7 Brake settings . . . 174
5.3.8 Position control . . . 176
5.3.9 Settings of external control devices . . . 177
5.3.10 General purpose input-output settings . . . 180
5.3.11 Motor type settings . . . 182
5.3.12 Settings of kinematics (DC motor) . . . 184
5.3.12.1 Motor parameters - electric motor settings . . . 184
5.3.12.2 Motion setup - settings related to the movement kinematics . . . 185
5.3.12.3 Feedback settings . . . 185
5.3.13 Settings of PID control loops . . . 185
5.3.14 About controller . . . 187
5.3.15 Settings of kinematics (BLDC motor) . . . 188
5.3.15.1 Motor parameters - electric motor settings . . . 189
5.3.15.2 Motion setup - settings related to the movement kinematics . . . 190
5.3.15.3 Feedback settings . . . 190
5.4 XILab application settings . . . 190
5.4.1 XILab general settings . . . 190
5.4.2 General motor settings . . . 192
5.4.3 Attenuator settings . . . 193
5.4.4 Cyclical motion settings . . . 194
5.4.5 Log settings . . . 195
5.4.6 Charts general settings . . . 196
5.4.7 Charts customization . . . 197
5.4.8 User units settings . . . 198
5.4.8.1 User units . . . 199
5.4.8.2 Coordinate correction table for more accurate positioning . . . 200
5.4.9 About the application . . . 200
5.5 Stage specifications . . . 201
5.5.1 Stage name . . . 201
5.5.2 Stage general characteristics . . . 202
5.5.3 Motor characteristics . . . 204
5.5.4 Encoder specifications . . . 206
5.5.5 Hall sensor characteristics . . . 208
5.5.6 Reducing gear specifications . . . 210
5.5.7 Accessories specifications . . . 211
5.6 Correct shutdown . . . 213
5.7 Working over network . . . 213
5.7.1 Getting started . . . 213
5.8 XILab installation . . . 215
5.8.1 Installation on Windows . . . 215
5.8.1.1 Installation on Windows XP . . . 215
5.8.1.2 Installation on Windows 7 . . . 224
5.8.1.3 Installation on Windows 8 . . . 231
5.8.1.4 Installation on Windows 10 . . . 234

5.8.1.5 Installation on Windows 11 . . . 237
5.8.2 Installation on Linux . . . 240
5.8.3 Installation on MacOS . . . 244

6 Programming **253**
6.1 Programming guide . . . 253
6.1.1 Working with controller in Visual Studio . . . 253
6.1.2 Working with controller in Delphi . . . 255
6.1.3 Working with controller in LabView . . . 257
6.1.4 Working with controller in Matlab . . . 260
6.1.5 Working with controller in ScanImage . . . 261
6.1.6 Working with controller in LabWindows . . . 261
6.1.7 A short description of the work with supported by programming languages . . . 263
6.1.7.1 Visual C++ . . . 264
6.1.7.2 .NET (C# and Visual Basic) . . . 264
6.1.7.3 Delphi . . . 264
6.1.7.4 Matlab . . . 264
6.1.7.5 Java . . . 264
6.1.7.6 Python . . . 265
6.2 Communication protocol specification . . . 266
6.2.1 Protocol description . . . 269
6.2.2 Command execution . . . 270
6.2.3 Controller-side error processing . . . 270
6.2.3.1 Wrong command or data . . . 270
6.2.3.2 CRC calculation . . . 270
6.2.3.3 Transmission errors . . . 271
6.2.3.4 Timeout resynchronization . . . 272
6.2.3.5 Zero byte resynchronization . . . 272
6.2.4 Library-side error processing . . . 272
6.2.4.1 Library return codes . . . 273
6.2.4.2 Zero byte synchronization procedure . . . 274
6.2.5 Controller error response types . . . 274
6.2.5.1 ERRC . . . 274
6.2.5.2 ERRD . . . 274
6.2.5.3 ERRV . . . 274
6.2.6 All controller commands . . . 274
6.2.6.1 Command GACC . . . 274
6.2.6.2 Command GBRK . . . 276
6.2.6.3 Command GCAL . . . 276
6.2.6.4 Command GCTL . . . 277
6.2.6.5 Command GCTP . . . 278
6.2.6.6 Command GEAS . . . 279
6.2.6.7 Command GEDS . . . 279
6.2.6.8 Command GEIO . . . 280
6.2.6.9 Command GEMF . . . 281
6.2.6.10 Command GENG . . . 282
6.2.6.11 Command GENI . . . 283
6.2.6.12 Command GENS . . . 284
6.2.6.13 Command GENT . . . 285
6.2.6.14 Command GEST . . . 285
6.2.6.15 Command GFBS . . . 286
6.2.6.16 Command GGRI . . . 286
6.2.6.17 Command GGRS . . . 287
6.2.6.18 Command GHOM . . . 287

6.2.6.19 Command GHSI . . . 289
6.2.6.20 Command GHSS . . . 289
6.2.6.21 Command GJOY . . . 289
6.2.6.22 Command GMOV . . . 291
6.2.6.23 Command GMTI . . . 292
6.2.6.24 Command GMTS . . . 292
6.2.6.25 Command GNME . . . 294
6.2.6.26 Command GNMF . . . 294
6.2.6.27 Command GNVM . . . 295
6.2.6.28 Command GPID . . . 295
6.2.6.29 Command GPWR . . . 296
6.2.6.30 Command GSEC . . . 296
6.2.6.31 Command GSNI . . . 297
6.2.6.32 Command GSNO . . . 298
6.2.6.33 Command GSTI . . . 299
6.2.6.34 Command GSTS . . . 300
6.2.6.35 Command GURT . . . 300
6.2.6.36 Command SACC . . . 301
6.2.6.37 Command SBRK . . . 302
6.2.6.38 Command SCAL . . . 303
6.2.6.39 Command SCTL . . . 303
6.2.6.40 Command SCTP . . . 304
6.2.6.41 Command SEAS . . . 305
6.2.6.42 Command SEDS . . . 306
6.2.6.43 Command SEIO . . . 307
6.2.6.44 Command SEMF . . . 307
6.2.6.45 Command SENG . . . 308
6.2.6.46 Command SENI . . . 310
6.2.6.47 Command SENS . . . 310
6.2.6.48 Command SENT . . . 311
6.2.6.49 Command SEST . . . 312
6.2.6.50 Command SFBS . . . 312
6.2.6.51 Command SGRI . . . 313
6.2.6.52 Command SGRS . . . 313
6.2.6.53 Command SHOM . . . 314
6.2.6.54 Command SHSI . . . 315
6.2.6.55 Command SHSS . . . 316
6.2.6.56 Command SJOY . . . 316
6.2.6.57 Command SMOV . . . 317
6.2.6.58 Command SMTI . . . 318
6.2.6.59 Command SMTS . . . 318
6.2.6.60 Command SNME . . . 320
6.2.6.61 Command SNMF . . . 320
6.2.6.62 Command SNVM . . . 321
6.2.6.63 Command SPID . . . 321
6.2.6.64 Command SPWR . . . 322
6.2.6.65 Command SSEC . . . 322
6.2.6.66 Command SSNI . . . 323
6.2.6.67 Command SSNO . . . 324
6.2.6.68 Command SSTI . . . 325
6.2.6.69 Command SSTS . . . 326
6.2.6.70 Command SURT . . . 326
6.2.6.71 Command ASIA . . . 327
6.2.6.72 Command CLFR . . . 327

6.2.6.73 Command CONN . . . 328
6.2.6.74 Command DBGR . . . 328
6.2.6.75 Command DBGW . . . 329
6.2.6.76 Command DISC . . . 329
6.2.6.77 Command EERD . . . 329
6.2.6.78 Command EESV . . . 330
6.2.6.79 Command GBLV . . . 330
6.2.6.80 Command GETC . . . 330
6.2.6.81 Command GETI . . . 331
6.2.6.82 Command GETM . . . 332
6.2.6.83 Command GETS . . . 332
6.2.6.84 Command GFWV . . . 335
6.2.6.85 Command GOFW . . . 336
6.2.6.86 Command GPOS . . . 336
6.2.6.87 Command GSER . . . 337
6.2.6.88 Command GUID . . . 337
6.2.6.89 Command HASF . . . 337
6.2.6.90 Command HOME . . . 338
6.2.6.91 Command IRND . . . 338
6.2.6.92 Command LEFT . . . 338
6.2.6.93 Command LOFT . . . 339
6.2.6.94 Command MOVE . . . 339
6.2.6.95 Command MOVR . . . 339
6.2.6.96 Command PWOF . . . 340
6.2.6.97 Command RDAN . . . 340
6.2.6.98 Command READ . . . 342
6.2.6.99 Command RERS . . . 342
6.2.6.100 Command REST . . . 342
6.2.6.101 Command RIGT . . . 342
6.2.6.102 Command SARS . . . 343
6.2.6.103 Command SAVE . . . 343
6.2.6.104 Command SPOS . . . 343
6.2.6.105 Command SSER . . . 344
6.2.6.106 Command SSTP . . . 344
6.2.6.107 Command STMS . . . 345
6.2.6.108 Command STOP . . . 345
6.2.6.109 Command UPDF . . . 345
6.2.6.110 Command WDAT . . . 346
6.2.6.111 Command WKEY . . . 346
6.2.6.112 Command ZERO . . . 346
6.3 8SMC1-USBhF software compatibility . . . 347
6.4 Libximc library timeouts . . . 349
6.5 XILab scripts . . . 349
6.5.1 Brief description of the language . . . 350
6.5.1.1 Data Types . . . 350
6.5.1.2 Statements . . . 351
6.5.1.3 Variable statements . . . 351
6.5.1.4 Reserved words . . . 351
6.5.1.5 Functions . . . 352
6.5.2 Syntax highlighting . . . 352
6.5.3 Additional XILab functions . . . 352
6.5.3.1 XILab log . . . 353
6.5.3.2 Script execution delay . . . 353
6.5.3.3 New axis object creation . . . 353

6.5.3.4 New file object creation . . . 354
6.5.3.5 Creation of calibration structure . . . 354
6.5.3.6 Get next serial . . . 355
6.5.3.7 Wait for stop . . . 355
6.5.3.8 libximc library functions . . . 355
6.5.4 Examples . . . 356
6.5.4.1 Bit mask example script . . . 356
6.5.4.2 A script which scans and writes data to the file . . . 357
6.5.4.3 Multi axis cyclic movement script . . . 358
6.5.4.4 Single axis cyclic movement script . . . 359
6.5.4.5 Homing test script . . . 360
6.5.4.6 List axis serials script . . . 361
6.5.4.7 Move and wait script . . . 361
6.5.4.8 Random shift script . . . 362
6.5.4.9 Set zero scrip . . . 363
6.5.4.10 Autotester script . . . 365
6.5.4.11 Border crossing test . . . 365
6.5.4.12 Closed loop tuning test . . . 367
6.5.4.13 Discrete motion script . . . 370
6.5.4.14 Exponential position change in user units script . . . 371
6.5.4.15 For calb step script . . . 374
6.5.4.16 Step script . . . 375
6.5.4.17 Homing test with extio . . . 375
6.5.4.18 Motion by sin function . . . 377
6.5.4.19 Move EXTIO calb script . . . 378
6.5.4.20 Probabilistic tests . . . 380
6.5.4.21 Several shifts with calibration script . . . 380
6.5.4.22 Steps loss test . . . 382
6.5.4.23 Sync test script . . . 383
6.5.4.24 Sync bug test script . . . 386
6.6 Community examples . . . 387
6.6.1 Example of six-axis XILab . . . 388
6.6.2 Examples for working with an attenuator for Python and LabView . . . 388
6.6.3 Program for sending commands to XIMC controller from AVR controller . . . 388
6.6.4 The multi-axis interface in Python . . . 388

7 Related products **390**
7.1 Control via Ethernet . . . 390
7.1.1 Ethernet adapters Overview . . . 390
7.1.1.1 General information . . . 390
7.1.1.2 Main requirements . . . 393
7.1.1.2.1 Network configuration . . . 393
7.1.1.2.2 Other . . . 394
7.1.2 Administration . . . 394
7.1.2.1 Automatic device detection . . . 395
7.1.2.2 Overview . . . 395
7.1.2.2.1 “Common” section . . . 397
7.1.2.2.2 “Motion control” section . . . 397
7.1.2.2.3 “Cameras” section . . . 398
7.1.2.3 Service control . . . 398
7.1.2.4 Firmware upgrade . . . 399
7.1.2.5 Troubleshooting . . . 400
7.1.2.5.1 Uninitialized Cubieboard2 bootloader . . . 400

8 FAQ **401**
8.1 No device found / Can't open device . . . 401
8.1.1 Connect via USB . . . 401
8.1.2 Connect via ETHERNET . . . 406
8.1.2.1 If the 8Eth1 adapter is not found on the local network . . . 406
8.1.2.2 The controller serial number is not displayed on the "Common" tab . . . 407
8.2 Unable to rotate the motor by the controller . . . 407
8.2.1 Controller has Alarm state . . . 408
8.2.2 Motor vibrates without rotation . . . 408
8.2.3 Mechanical jamming . . . 412
8.2.4 The motor does not react on move commands . . . 412
8.3 Self-test of the 8Eth1 adapter . . . 412
8.3.1 Instructions for installing a static ip image . . . 413
8.3.2 Static ip address self-test on 8Eth1 . . . 413
8.4 When using the libximc library and Linux with kernel version less than 3.16, there are possible hanging of the operating system . . . 415
8.5 USB connection loss . . . 415
8.6 *probe_flag* - what is it? . . . 416
8.7 Virtual controller as in XILab Software . . . 416
8.8 Python CRC algorithm . . . 418
8.9 Where can I find the programming manual for the 8smc5 controller? . . . 418
8.10 How do I implement an emergency stop button? . . . 418
8.11 How to get a xilab window that has disappeared off the screen? . . . 419
8.12 How to check if the connection to 8SMC5-USB is established and still active during my session using the libximc library? . . . 420
8.13 The problem of compensation for backlash (example from technical support) . . . 420

8SMC4/5 technical support

- Ask a question
- Send an e-mail: **8smc4@standa.lt**
- For quick simple questions contact us:
- **via Telegram**: @SMC5TechSupport
- **via WhatsApp**: +1 (585) 282-6387
- *Available online Mon-Fri 8:00-16:00 UTC*

For controllers purchase

- On the purchase of controllers, please contact by e-mail: **sales@standa.lt**

CHAPTER

ONE

ABOUT

1.1 General information

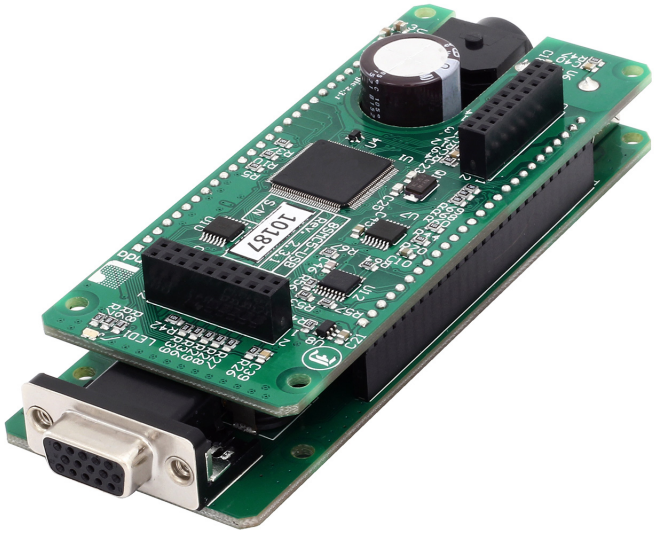
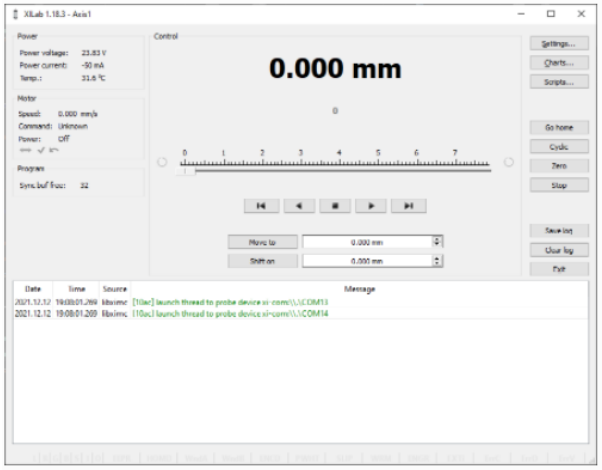


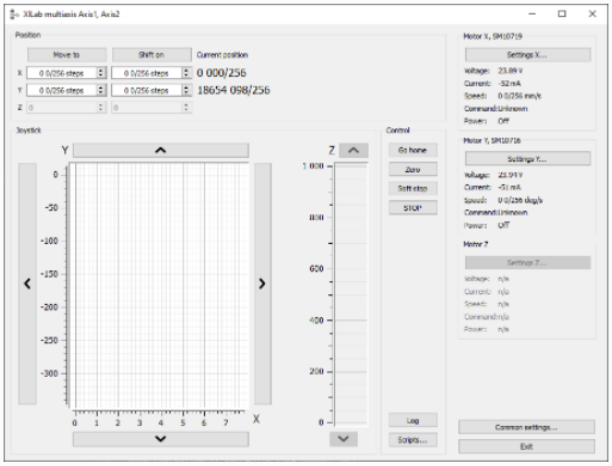
Fig. 1.1: Controller board without case

We offer an inexpensive and ultra-compact servo-drive with USB interface for stepper, DC and BLDC motors with external power supply.

Forget about cumbersome and expensive servo-drives! All you need is a stepper, DC or BLDC motor, a controller, an USB cable and any stabilized external power supply. That is all! Forget about active coolers as well. With modern controller design even simple and inexpensive stages can be utilized to achieve high speed and precision. It doesn't matter which motor technology you prefer, because one controller can drive them all. The controller is great at driving stepper motors with a rated winding current of up to 3A and DC motors with rated current up to 6A. Controller works with stepper motors with no feedback as well as with ones equipped with encoders in feedback loop, including linear encoders on the stages. The motor connector on the controller board is the same as one used by Standa company and it fits to all the Standa stages. USB connector provides easy communication and work with computer. Several controllers can be connected to one computer either via USB ports or through a special backplane supplied with multiaxis systems. The controller is fully compatible with the majority of operating systems, e.g., Windows, Mac OS X, Linux, etc.



XILab main window (Single Axis mode)



XILab main window (Multi Axis mode)

All the necessary software including all configuration files (profiles in .cfg format) are supplied with the controller. It allows you to start working with it the controller right out of the box, according to “plug-and-play” principle.|br| Therefore, all you need is to open your controller in the XILab software, download the .cfg file for your stage and click “Apply”. Your controller is now fully configured! Enter the move commands and the controller will execute them.

1.2 Benefits

1.2.1 Main benefits

- *Compact and powerful!* The controller’s dimensions are 127 x 46 x 32 mm including all connectors. The device is adapted to all stepper motors with rated winding current of up to 3A.
- *Compatible with 8SMC1-USBh* After updating the MicroSMC driver, all the software for 8SMC1-USBh will be working with 8SMC5-USB.
- Compatible with all the Standa stages! Have you got a Standa stage? Just plug and play, we have already done the rest for you!
- *It does remember all!* Do not worry about saving the current position on the computer: the controller does it itself using its own nonvolatile memory that works even after a sudden power cut.
- It works with peripherals! It supports a *quadrature encoder*, *magnetic brake*, a *joystick*, *limit switches*, a zero position sensor – all included, just plug and work!
- *Built-in zero calibration!* Using the *limit switches*, the *revolution sensor*, the *external signal* or their combination, the zero calibration is performed by a single *command*!

1.2.2 All benefits

- Really powerful! The controller is great at driving stepper motors with a rated winding current of up to 3A and DC motors with rated current up to 6A.
- *Compatible with 8SMC1-USBh* After updating the MicroSMC driver, all the software for 8SMC1-USBh will be working with 8SMC5-USB.
- Compatible with all the Standa stages! Have you got a Standa stage? Just plug and play, we have already done the rest for you!
- *It knows its own set!* A built-in feature for downloading the configuration file right from the stage memory is available! The stage’s support of such memory is required.

- Choose your interface! Both USB and *serial port* are built-in and ready to use.
- Really fast! Up to 35,000 steps per second for any microstep mode!
- Precise! The microstep modes from full step to 1/256 of the step on all the speeds.
- *It does remember all!* Do not worry about saving the current position on the computer: the controller does it itself using its own nonvolatile memory that works even after a sudden power cut.
- It works with peripherals! It supports a *quadrature encoder*, *magnetic brake*, a *joystick*, *limit switches*, a zero position sensor – all included, just plug and work! Additional stabilized output for peripherals (5V, 500mA) is available.
- *Built-in zero calibration!* Using the *limit switches*, the *revolution sensor*, the *external signal* or their combination, the zero calibration is performed by a single *command*!
- Stand-alone! Would you like to work in the stand-alone mode? Just go ahead! An external *joystick*, a *keypad* or their combination is supported.
- *Energy conserving!* Programmable current reduction in the motor windings in the hold mode with 1% accuracy.
- Silent! Smooth movement at lower speeds and no extra noise at higher speeds.
- Protected! An ESD protection on all pins of external connectors and additional short circuit protection for the motor windings.
- *Attentive!* It controls the temperature of the processor and the power driver as well as both currents and voltages for the power supply and USB.
- Modern! The firmware in the nonvolatile memory of the controller *can be updated* via USB interface.
- Controlling and controllable! The built-in *synch input and output* allow to start the rotation to desired position by the incoming external signal and/or to transmit the outgoing signal after the desired position is reached. The *analog common input* and the *digital common input/output* are built in as well.
- Comprehensible! The *status LED* displays the power supply and the controller's state. For convenience of use both signals doubled at the external LEDs as well as the state of the limit switches.
- *It works with all computers!* All the supplied software is compatible with *Windows 11*, *Windows 10*, *Windows 8*, *Windows 7*, Windows Vista, *Windows XP SP3*, *Linux*, *Mac OS for intel*, including 64-bit versions and Apple Silicone (via Rosetta 2).
- Examples for all programming languages! Controllers are supplied with cross-platform library and examples which allow rapid development using C++, C#, .NET, Delphi, Visual Basic, Xcode, Python, Matlab, Java, LabWindows and LabVIEW.
- *Full-featured interface!* The XILab user interface is supplied with the controller. It allows to easily control all the functions and features of the device without any programming.
- *Unique scripting language!* A scripting language is integrated into XILab software. It allows easy setting the sequence of actions, including cycles and branches, without compilation or learning any programming language.
- *Stepper motor close-loop control algorithm are ready!* Motion is smoother and faster than ever with innovation encoder based close-loop on 8SMC5-USB motor controllers. No hidden catch, no stall or hitch, just free move!

1.3 Table of specifications

Power supply	external 12 V - 48 V DC
Current consumption	up to 5 A (depending on the voltage) from external power supply

Continued on next page

Table 1.1 – continued from previous page

Current in the stepper motor, BLDC motor winding	up to 3 A		
Current in the DC motor winding	up to 6 A		
Protection types	current overload protection, voltage overload protection, short circuit protection, motor hotplug/unplug protection		
Motion modes	move left/right, move to point, shift on delta, continuous speed, acceleration and deceleration ramps, backlash compensation mode, automatic home position calibration mode		
Step modes	full-step, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128, 1/256		
Rotation speed	up to 35 000 full steps per second		
Speed profile	trapezoidal		
Position counter	40 bit		
Feedback	Single-ended and differential quadrature encoder (optional)		
Limit switches	2		
Feedback bandwidth	200 kHz for single-ended and 5MHz for differential encoder		
Sampling rate of the sync output signal	1 ms		
Controller weights	8SMC5	8SMC5-USB-B8-1	8SMC5-USB-B9-2
	93 g	500 g	621 g
Controller dimensions	*46x126,75x31,81 mm*	*66x46x114 mm*	*122x45x106 mm*

Note: The controller's working voltage range is 12 V to 48 V DC. The voltage limits are 12V and 50 V DC. If the voltage exceeds 50 V, the controller is guaranteed to fail. If the voltage falls below 12 V, the controller turns off.

1.4 Specifications

1.4.1 Motor requirements

- Motor type: bipolar stepper motor, DC, BLDC motor.
- Rated winding current: minimum 100 mA.
- Rated winding voltage: minimum 2 V DC.

1.4.2 Electric specifications of the controller

- Power supply modes: external power supply.
- Current in each winding of the stepper motor, BLDC motor: up to 3 A.
- Current in each winding of the DC motor: up to 6 A
- Maximum encoder pulse frequency: 200 kHz for single-ended and 5MHz for differential encoder.
- Stabilized 5V DC output (the power supply for encoder and other peripherals): 500mA maximum output current, 5% or better output voltage stability.

- ESD-protection on all pins of the output connectors (e.g., D-Sub 15 pin, mini-USB or power jack).
- Winding-to-ground short circuit protection.
- Winding-to-winding short circuit protection.
- Motor hot-swapping protection.
- Wrong power polarity protection (no more than 1 sec).
- Voltage overload protection (no more than 1 sec).
- USB-supplied current limitation.
- External power supply current limitation.
- Motor rotation speed limitation.
- Programmable full winding current with 10mA precision.
- Programmable winding current decrease with 1% precision for the hold mode.
- For earth conditions, the temperature range of the controller is from +5 °C to +75 °C.

Note: The controller was not tested in a vacuum. Most likely, the controller will work in a vacuum, but it is important how the heat will be removed from the body.

1.4.3 Rotation control features

- Microstep modes: full-step, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128, 1/256.
- Noiseless at low speeds.
- Minimum speed is 1/256 of the full step per second.
- Maximum speed is up to 35 000 full steps per second for all microstep modes.
- Minimum shift is 1/256 of the step.
- Maximum shift is 2,147,483,647 full steps for all microstep modes.
- Smooth start/stop mode.
- 40-bit position counter (32 bits for full step and 8 bits for microstep).
- Motion modes: left/right move, move to point, shift on delta, continuous speed, acceleration and deceleration ramps, backlash compensation mode, automatic home position calibration mode.

1.4.4 Additional firmware features

- Automatic HOME calibration at firmware level.
- The nonvolatile memory used for saving/downloading the controller settings.
- Software update via USB interface.
- Automatic position saving according to step counter and encoder with power-off protection.

1.4.5 Additional features available via motor connector

- Processing the signals from one or two limit switches; software configurable.
- The Standa stages recognition and automatic downloading of the configuration file right from the stage if the last one supports this feature.

- The “step loss” detection and position recovery using either a revolution sensor or a quadrature encoder (if the stage supports this feature).
- The position detection using a quadrature encoder. The x4 mode.
- The stepper motor control using master quadrature encoder mode, providing the maximum speed without any step loss. Starting with firmware 4.1.

1.4.6 Additional features available via backplane connector

- USB connector on backplane that duplicates USB input on controller board.
- A serial RS-232 port. TX and RX lines are available. Specifications: 9600 - 921600 baud speed, TTL 3.3V. The Ethernet, Bluetooth, WiFi, ZigBee and other configurations based on the serial port are available by request.
- Synchronization input: once the pulse is received via this pin, the controller starts rotating the motor to predetermined position or by predetermined shift value. The triggering mode, the polarity and duration of the pulse are adjustable by user. Specifications: TTL 3.3V.
- Synchronization output: emit pulse to this pin if rotation is started or finished, or predetermined user-defined shift value is reached. The triggering mode, the polarity and duration of the pulse are adjustable by user. Specifications: TTL 3.3 V.
- Left or right buttons. Once the button is pressed, the rotation in corresponding direction starts and the speed increases gradually according to acceleration and other settings. Specifications: TTL 3.3 V.
- Joystick pin allowing operation with various joysticks with the voltage range no more than 0–3 V.
- Magnetic brake control pin providing control to magnetic brake mounted on the motor shaft. Specifications: TTL 3.3V, 5mA.
- Common analog input pin allowing operation with signals within 0–3 V range. Reading frequency is 1kHz. The configuration is programmable.
- Common digital input/output pin. 1kHz update frequency, software configurable. Specifications: TTL 3.3V, 5mA.
- Limit switches indication pins designed for LED direct connection. Specifications: TTL 3.3 V, 2 mA.
- Digital “Power” and “Status” pins duplicate the status LED and designed for direct connection of LEDs. Specifications: TTL 3.3 V, 2 mA.
- External driver control interface allowing to control any type of external driver using three signals: enable, direction, clock.
- Multiaxis systems development. The multiaxis systems are created from standard USB hubs, either external or mounted at a special backplane. On the PC a multiaxis system is represented as a set of virtual serial ports, according to the number of connected axes.

1.4.7 Programming the controller

- All the software supplied with controller is compatible with Windows 11, Windows 10, Windows 8, Windows 7, Windows Vista, Windows XP SP3, Linux, Mac OS for intel and Apple Silicone (via Rosetta 2) including 64-bit versions.
- Controllers are supplied with cross-platform library and examples which allow rapid development using C++, C#, .NET, Delphi, Visual Basic, Xcode, Python, Matlab, Java, LabWindows and LabVIEW.
- The XILab user interface is supplied with the controller. It allows to easily control all the functions and features of the device without any programming.

- A scripting language, an EcmaScript language dialect, is integrated into XILab software. It allows easy setting the sequence of actions, including cycles and branches, without compilation or learning any programming language.

Attention: The programming guide can be found on our second site libximc.xisupport.com or you can download a PDF version of it

CHAPTER

TWO

SAFETY INSTRUCTIONS

2.1 Power supply and grounding requirements. Connection to controller

General requirements for a *controller board*, systems in box (*single* and *double* axes) are listed below.

During operation, current consumption will vary depending upon how the controller is being used. Our controllers are calibrated to the rated current of the motors they are to be used with. Due to Pulse Width Modulation (PWM) our controllers consume less current than the rated current of motors. However, to avoid problems during worst case scenarios, we recommend selecting a power supply with a max current not less than the rated current of motors that will be connected to the controller. In case of multi-axis controllers you will need to sum the current of all controllers connected to the power supply. **Our controllers require a voltage of 12V-48V. Recommended power supply parameters: 24V; 2.5A**

Important: Either the power supply unit should be plugged to grounded 220V AC socket (a three-wire connection scheme). Make sure that the minus electrode of your power supply unit is grounded. If the power supply unit with minus electrode disconnected from the grounding circuit is used, ground the **controller board** via special **grounding terminal** (look at the picture below). If controller is supplied inside the metal case, the case must be grounded. Non-compliance with these rules may lead to the decrease in controller stability and noise resistance.

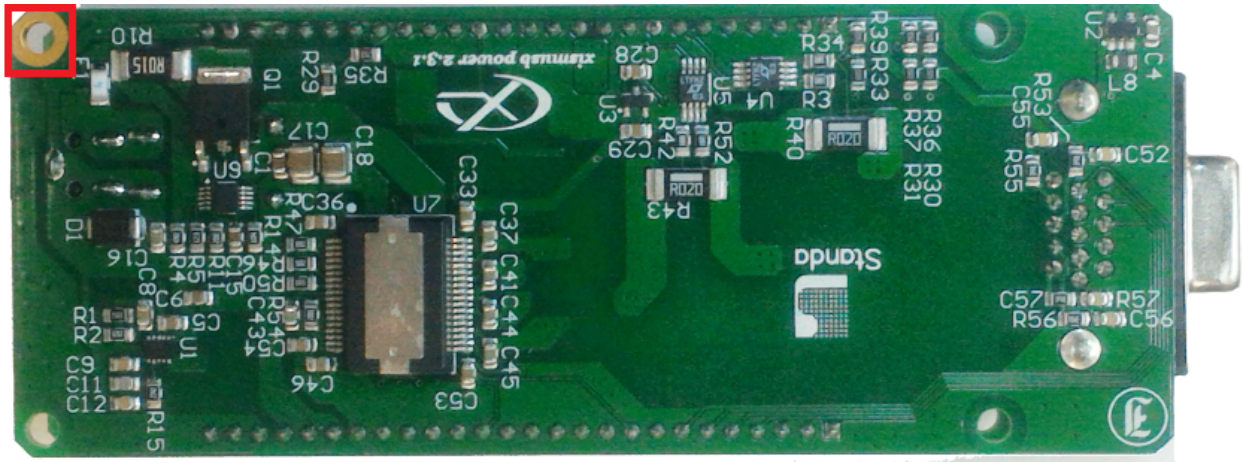


Fig. 2.1: Top view of the controller board. A grounding terminal marked with the red square

Typical connection diagram for a controller (board, systems in box):

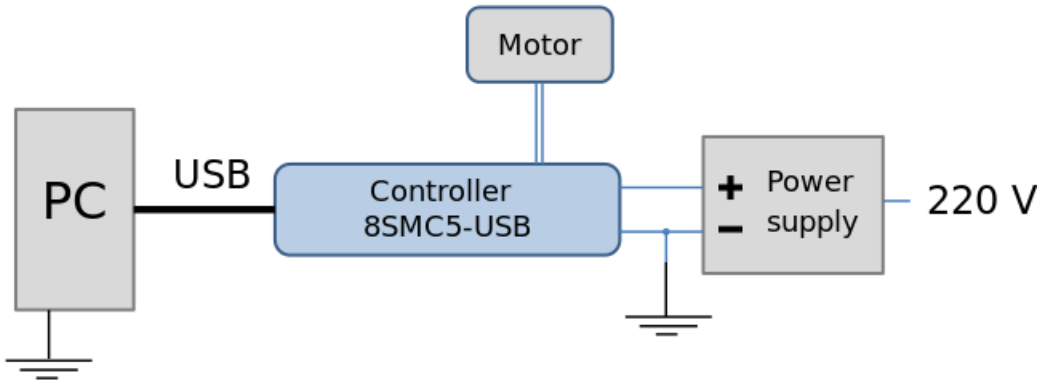


Fig. 2.2: Controller grounded via minus electrode of power cable connection diagram

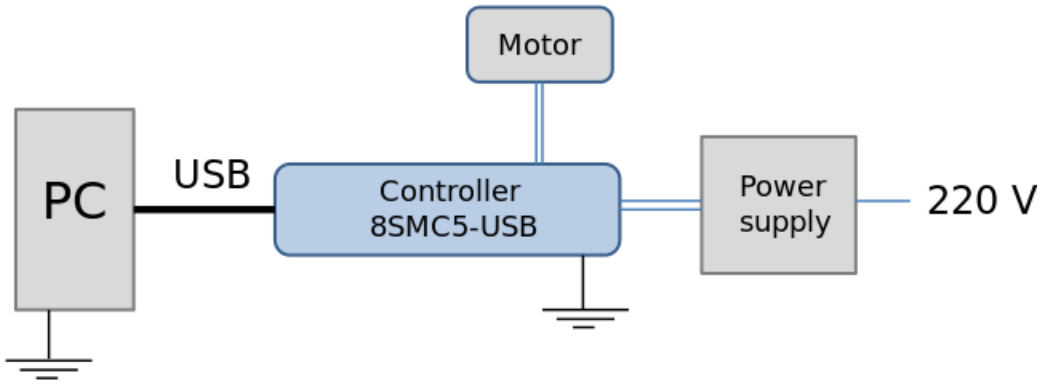


Fig. 2.3: Controller connection diagram with grounding via special terminal

Warning: Power supply unit should be able to supply sufficient current to rotate the motor. As an absolute minimum it should be able to supply

$$I_{power.min} = \frac{2 * I_{motor} * U_{motor}}{U_{power}}$$

where $I_{power.min}$ is the minimum working current of power supply unit, I_{motor} is the operating current in the winding, U_{power} is power supply unit stabilized voltage, and U_{motor} is rated operating voltage of the motor. It is recommended to use a power supply unit with operating current equal to $I_{power} \geq 2 * I_{power.min}$. The U_{power} voltage should be greater than U_{motor}. The higher the voltage, the faster rotation speed could be reached.

One can use power consumption of power supply unit to calculate minimum requirements instead. An absolute minimum of power is:

$$W_{power.min} = I_{power.min} * U_{power} = 2 * I_{motor} * U_{motor}$$

For example, for motor with operating winding current of 1A and operating voltage of 5V (with 5W rated power consumption), the operating voltage of power supply unit may be chosen at 20V with the output power of at least 10W (the maximum operating current of power supply unit is at least 0.5A).

2.2 Controller board

Important: It is strictly forbidden to touch the *controller board* without any antistatic equipment. We recommend you to use antistatic wrist strap. **You should not exceed maximum allowed voltage of 48V**. If voltage goes over allowed value at more than 2 volts, it can immediately and irreversibly damage the controller.

Important: It is strictly forbidden to connect positive wire of power supply to the *controller board* when ground wire is not connected. It is strictly forbidden to connect or disconnect power cable when the power supply is on, the controller is connected to the PC and the power supply and the PC are grounded. **This may damage the PC!** This is a common requirement for any electronic device with separate power source.

Warning: Before connecting the *controller board* to the motor or to the PC via USB interface, it is recommended to connect controller to power supply with proper grounding or to use separate grounding for controller with the specially marked ground terminal (look *above*).

2.3 One and two axes system in box

Important: **You should not exceed maximum allowed voltage of 48V**. If voltage goes over allowed value at more than 2 volts, it can immediately and irreversibly damage the controller.

CHAPTER

THREE

QUICK START GUIDE

This guide describes the operation of controller for one-axis systems, basic parameters configuration and getting started with XILab software for Windows 7.

Attention: For a quick start with the controller, see the *Overview and getting started*. The programming guide can be found on our second site libximc.xisupport.com or you can download a PDF version of it

- *Overview and getting started* - a brief description of the beginning of work with the controller 8SMC5-USB for one axis. It is also considered quick XiLab setup and lists all necessary equipment.
- *Multiaxis system* - a brief description of the beginning of work with multi-axis system based on the 8SMC5-USB controller. It is also considered XiLab setup, lists all necessary equipment and describes characteristics of the system.
- *Example of a motor connection* - connection of stepper motor Nanotec ST5918L3008-B with encoder CUI INC AMT112S-V to 8SMC5-USB controller. It is described how to make your own cable, guided by the specification on the engine and explanation of the specification is given.
- *Manual profile setting* - setting of working profile for XiLab. Overview of the main features.
- *Calculation of the nominal current* - setting of amplitude of nominal current for stepper motors.

3.1 Overview and getting started

- *Introduction*
- *Requirements*
- *Software installation and startup procedures*
- *Getting started with XILab software*
- *Functional test*
- *Control from user applications*

Attention: This manual is universal for both uniaxial and biaxial 8smc5 controllers

https://youtu.be/LIU633PFQ94

Video of a quick start with the controller connected via USB (for windows)

https://youtu.be/OmqcL8k9Feg

Video of a quick start with the controller connected via Ethernet (for windows)

3.1.1 Introduction

This manual describes the controller installation procedures and getting started with XILab software for Windows 7. The installation on other OSs is described in *XILab installation* chapter. The detailed controller specifications are described in *Specifications* chapter. For developing your own applications, please read the *Programming guide* chapter and download the programming software package from the software chapter.

3.1.2 Requirements

For successful installation you will need:

- **PC with USB port**



- **Software** All necessary software to work with the controller can be downloaded from software page.
- **USB Type-A - USB Type-B cable**



- **8SMC5-USB**, **8SMC5-USB-B8-1** or **8SMC5-USB-B9-2** controller

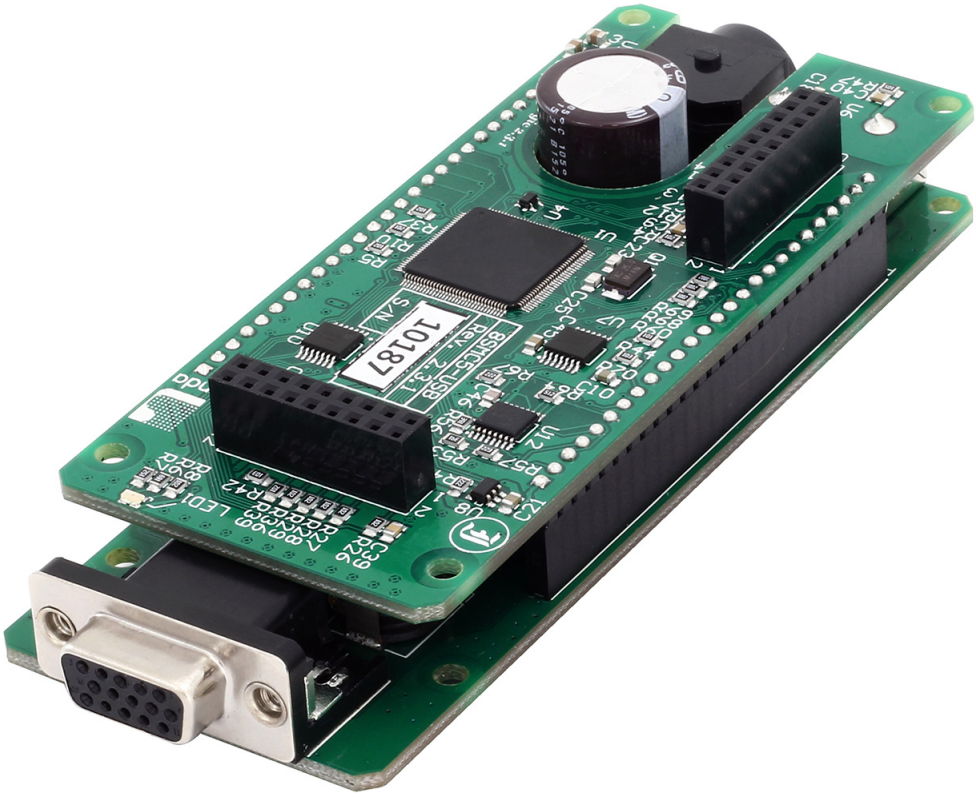


Fig. 3.1: 8SMC5-USB Controller without case



Fig. 3.2: 8SMC5-USB-B8-1 One Axis Controller



Fig. 3.3: 8SMC5-USB-B9-2 Two Axis Controller

- **Stage or motor**



Fig. 3.4: The stepper motor-based stage

The stepper motor-based stage used in the operations is shown at the figure. The detailed motor requirements are described in *Specifications* chapter. If you use your own cables for connecting the stage to the controller, please refer to *stage connection scheme* and *the controller's output connector scheme*. For stages with limited movement range, two *limit switches* must be used: SW1 and SW2. These pins are used to determine the movement limits.

- **Power supply**



- Please use the 12-36V DC stabilized power supply. Too high voltage may damage the controller. For more information please read the *Safety instructions* chapter. The power supply unit must provide the current enough for sustainable rotation of the motor.
- If controller is supplied inside the metal case, the case must be grounded. If controller is supplied without any body, the grounding circuit of power supply unit is used. For more information please read the *Safety instructions* chapter.
- If the board is operated without the casing, make sure it lays on the insulating surface.

3.1.3 Software installation and startup procedures

You can download the software here. The installer file name is "xilab-<version_name>.exe". It automatically detects whether it is running on 32-bit or 64-bit version of Windows and installs the appropriate version of XiLab. Launch the installation program, the installation window will appear. (The software versions may slightly differ from each other).



Fig. 3.5: XILab main installation window

Press "Next>" button and follow the instructions on screen. All the necessary software including all drivers, packages and programs will be installed automatically. After installation is finished, the XiLab software starts by default and the following window will open:

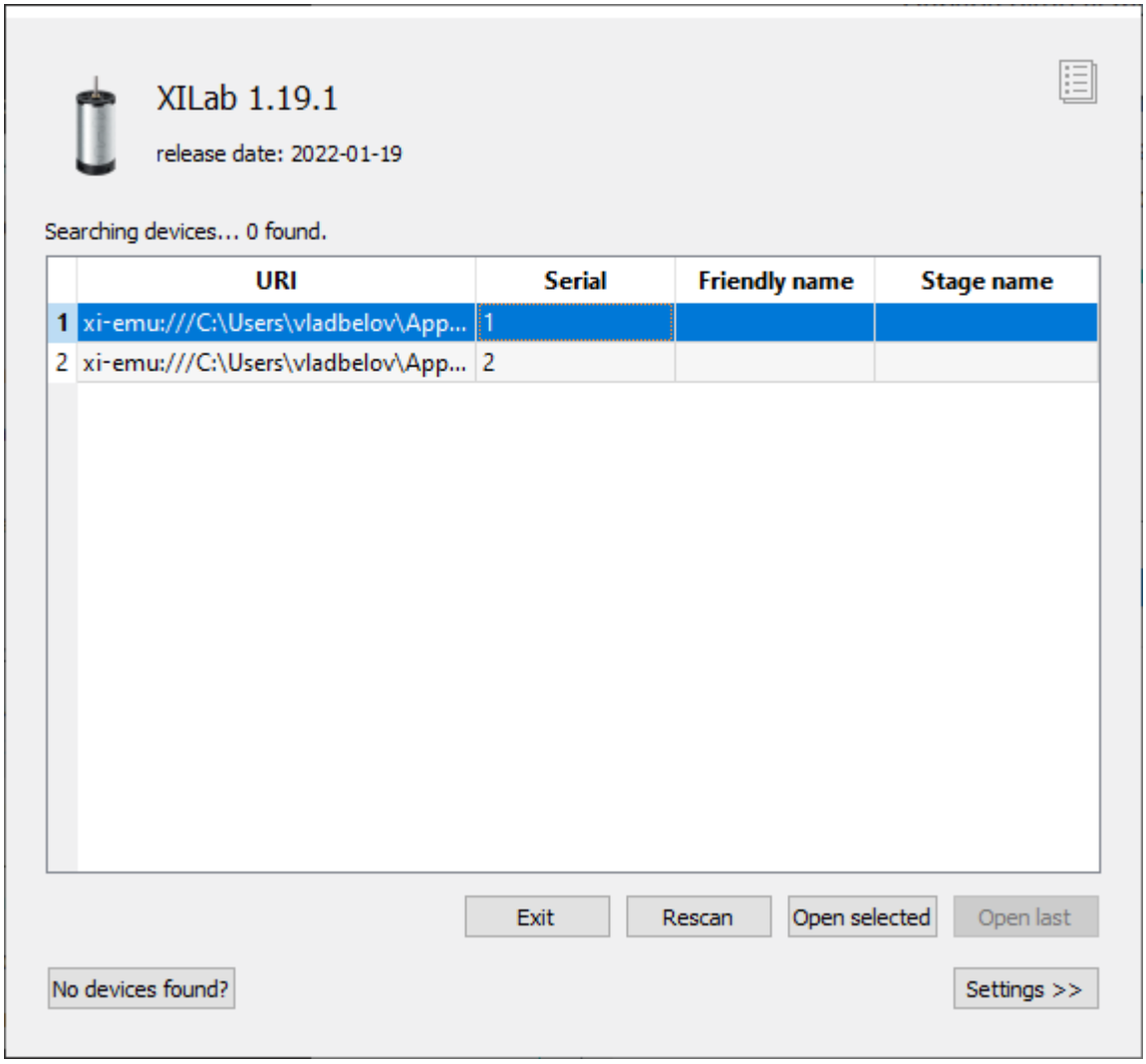


Fig. 3.6: XILab "Virtual controllers found" dialogue window

Don't press any buttons. Connect the stage to the controller. Connect the stabilized power supply unit to the controller. Ground the controller or power supply unit. Connect the controller to your PC using the USB Type-A to USB Type-B cable.

The LED indicator at the controller board will start *flashing*. The New Hardware Wizard starts working after the first connection of the controller to PC. Please wait until Windows detects a new device and installs all necessary drivers for it.

If the automatic driver installation has failed, please select "No, not this time" in the window being opened and press "Next>" button. Select "Install from a list or specific location (Advanced)" in the next window and press "Next>" again. Browse the software disk supplied with controller and find the **.inf* file there or in the **XiLab-install-path\driver** folder and wait until installation is completed.

Go back to XILab "Virtual controllers found" dialog window and press "Rescan" button. If this window was closed, please reopen XILab software. The dialog window will open again.

3.1.4 Getting started with XILab software

XILab is a user-friendly graphic interface designed for control, diagnostics and adjustment of motors. It can also be

used for easy installation and save/restore of parameters for any type of motors. This chapter describes the startup procedures with XILab software. For complete information please refer to *XILab application User's guide* chapter.

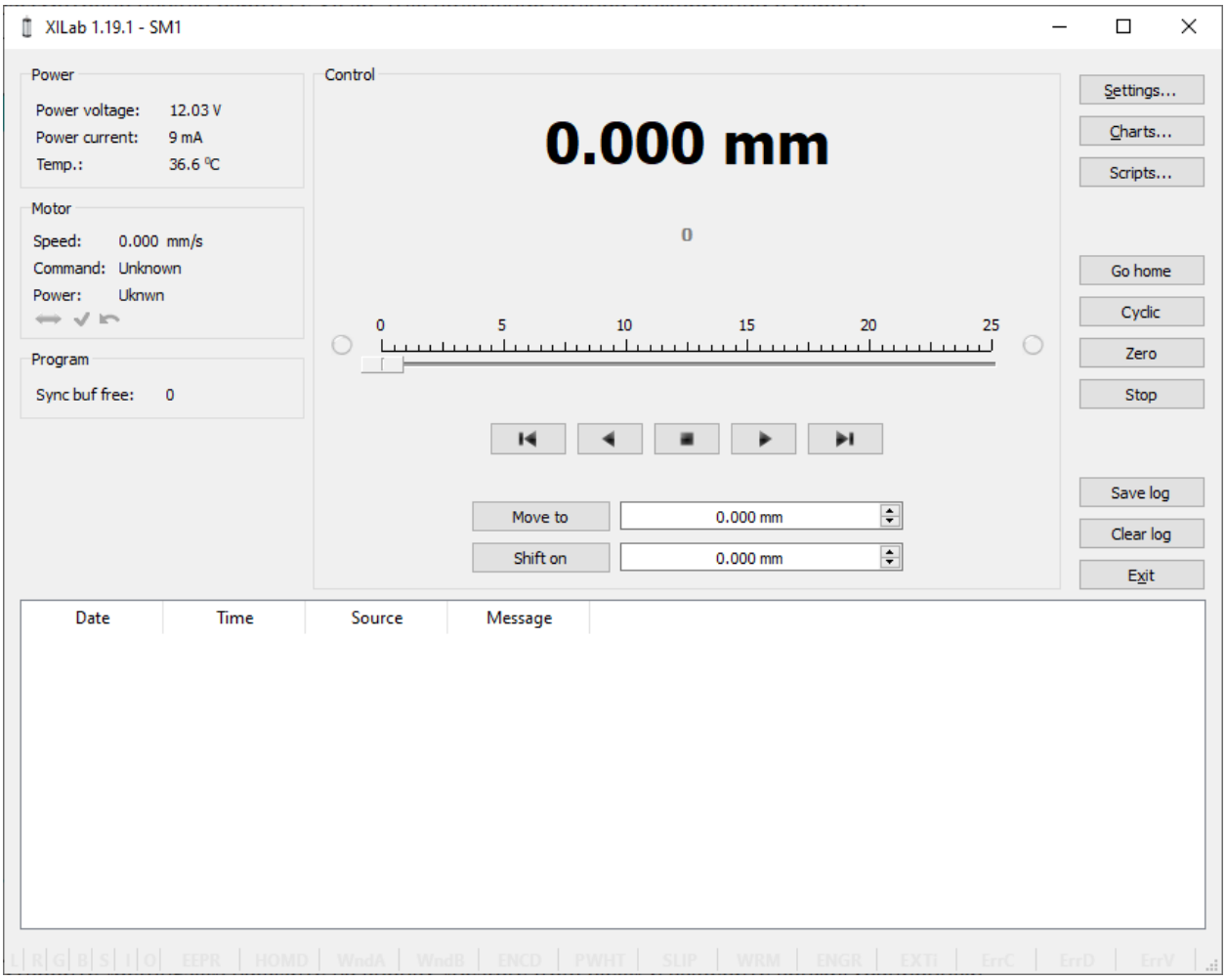


Fig. 3.7: XILab main window

Open "Settings…", then press "Load setting from file…" and select the configuration file for your stage from the opened **C:\Program Files\XILab\profiles** folder. The values applicable for your stage will automatically fill all the fields of "Settings…" menu. If the necessary file isn't found, please refer to the Standa website. If there is still no solution, please leave your request at our technical support website.

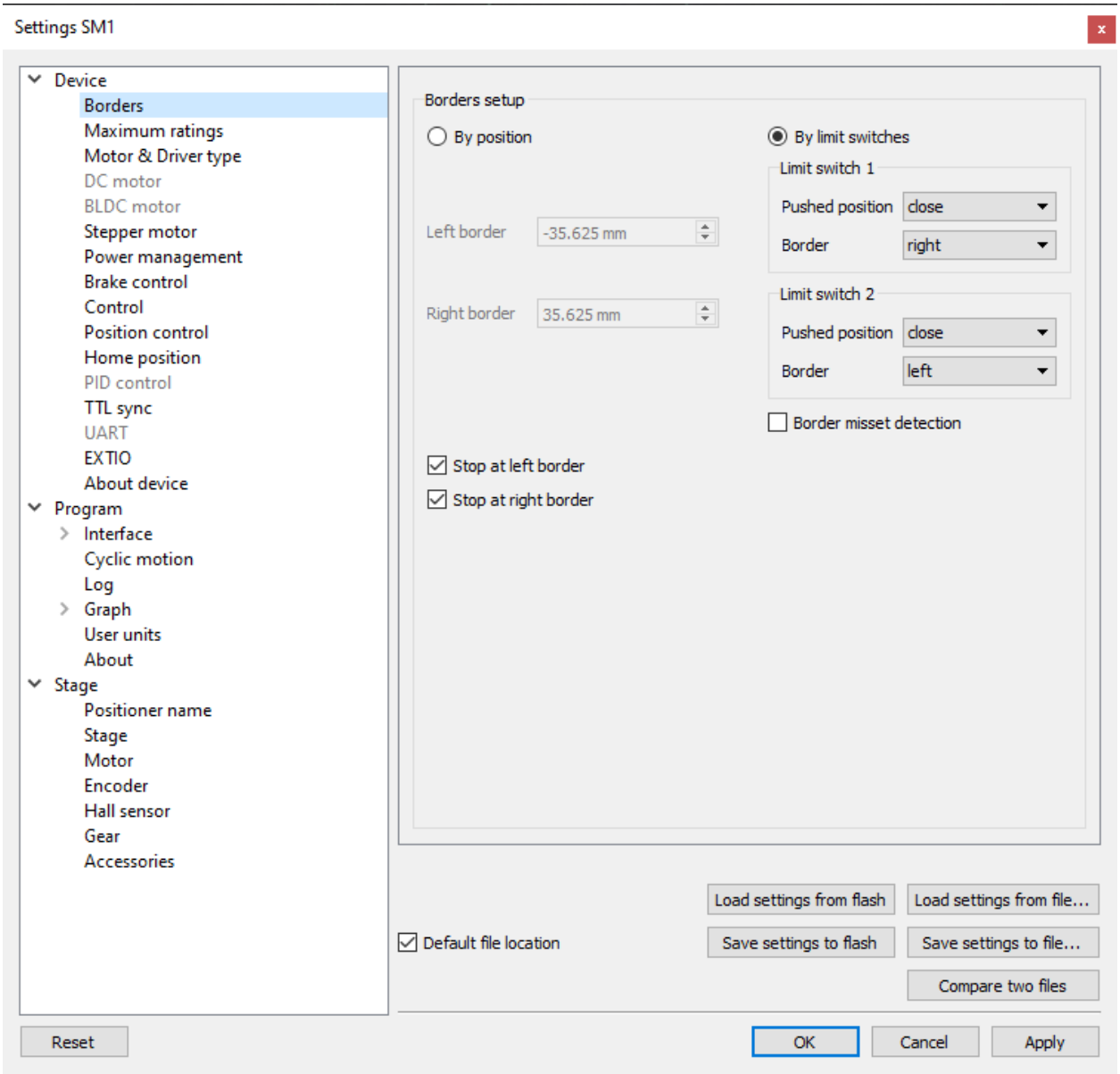


Fig. 3.8: XILab, the Settings menu window

Warning: For the controller to work with motors it is required to properly set up:

- *working current*,
- *displacement limits and limit switches*,
- *critical parameters*,
- *limiters*,
- *power supply mode*.

If you decide to configure your controller by yourself, please check these parameters carefully!

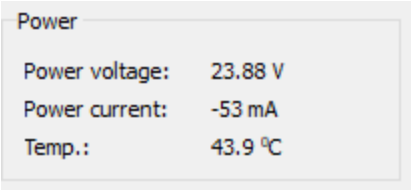
Congratulations, your controller is ready for operation!

3.1.5 Functional test

Check if the controller is configured properly by pressing left or right button in the central row of XILab main window control buttons. The stage has to start moving. Use the central "soft stop" button to stop the rotation.



Please pay attention to the power supply parameters of the controller in the *Power* section. There you can see the power voltage, working current and temperature of the controller.



If Xilab main window is shaded red when the movement was supposed to start, that means that protection was activated and controller entered the *ALARM* state. This may be caused by incorrect settings, wrong connection of the stage or controller malfunction. For detailed information please read the *Critical parameters* chapter.

3.1.6 Control from user applications

For convenient control of the 8SMC5 controller, you can use the *Xilab software*. However, if you need to control the 8SMC5-USB from your own application, you may do so by using libximc library. *Programming guide* has several examples in *C*, *C#, VB.net*, *Python*, *Delphi*, *Java*, *Matlab*, *LabView* programming languages. If all you need is to automate a small number of control steps, then instead of a standalone program you may find it easier to use Xilab *scripting language*.

3.2 Multiaxis system

- *Introduction*
- *Required components*
- *Getting started with multi-axis XiLab.*
- *Multi-axis system temperature control.*
- *Errata*
  - *Operation*
  - *Construction*



Fig. 3.9: Appearance of the multi-axis system

3.2.1 Introduction

This manual describes start with multi-axis system (8SMC4-36) based on 8SMC4-USB motor controller. 8SMC4-USB getting started descibed in *overview and getting started* chapter. Multi-axis system operates from 220 V and it allows to control 36 axis simultaneously.

3.2.2 Required components



Fig. 3.10: 220 V power cable



Fig. 3.11: Ethernet cable. It allows you to control multi-axis system remotely

DHCP server is needed for control of multi-axis system via local network. You should know ip address of your devices in network.

3.2.3 Getting started with multi-axis XiLab.

Connect 220V and Ethernet cable to multi-axis system. Press Power On/Off button at the back panel. After 2 minutes 8SMC4-B36 is ready to use. It needs time to start operating system on board computer for multi-axis system control.

At start you need XILab with multi-axis support version 1.13.x or newer. Link to download. Setup process and getting started with XILab is described *here*. For complete information please refer to *XILab application User's guide*.

The setting up access to remote controllers via Ethernet is described in *Working over network* chapter. In controller detection window choose an axis you need. You can control it in *single-axis mode* or in *multi-axis mode* if more than one axis was chosen. For additional information please refer to *Getting started with XILab software* and *XILab application User's guide*

3.2.4 Multi-axis system temperature control.

Multi-axis system 8SMC4-36 is able to control heating of its controller's during operation. It can measure temperature of circulating air via thermal resistors. With temperature increase speed of cooling system fans rotating will increase too.

At temperature of circulation air about 25^oC cooling system is at 10% of its maximal power. 100% of power achieves with temperature about 25^oC.

3.2.5 Errata

All of the following notes are for a multi-axis system version 1.1.

3.2.5.1 Operation

Main problems related to multi-axis system operation:

- When powered from 110V full power is limited to 1 kW;
- While simultaneous emergency stop of several axes occurs at high power motors, there can be short-term connection loss with a multi-axis system control interface;
- There is possible a failure with consiquent axes rediscovery in OS due to modules hot plug;
- When updating the firmware, controllers can lose USB connection, and are not available in XiLab, until the system is restarted;
- It is possible that when inserting the module in the operating system, it is not energized and fails to work.

3.2.5.2 Construction

Structural defects of the multi-axis system that may be encountered during disassembly and assembly:

- ic9xbb board. Silkscreen P1 - P5 must be on the other side of the corresponding connectors;
- ic9xbb board. Mixed up silkscreen of P1 and P3 connectors.

3.3 Example of a motor connection

- *General case*
- *Example*

- *Preparation*
- *Connecting the motor and encoder to the controller*

3.3.1 General case

To connect a motor to the controller please refer to *stage connector*, and use the scheme of stage connection:

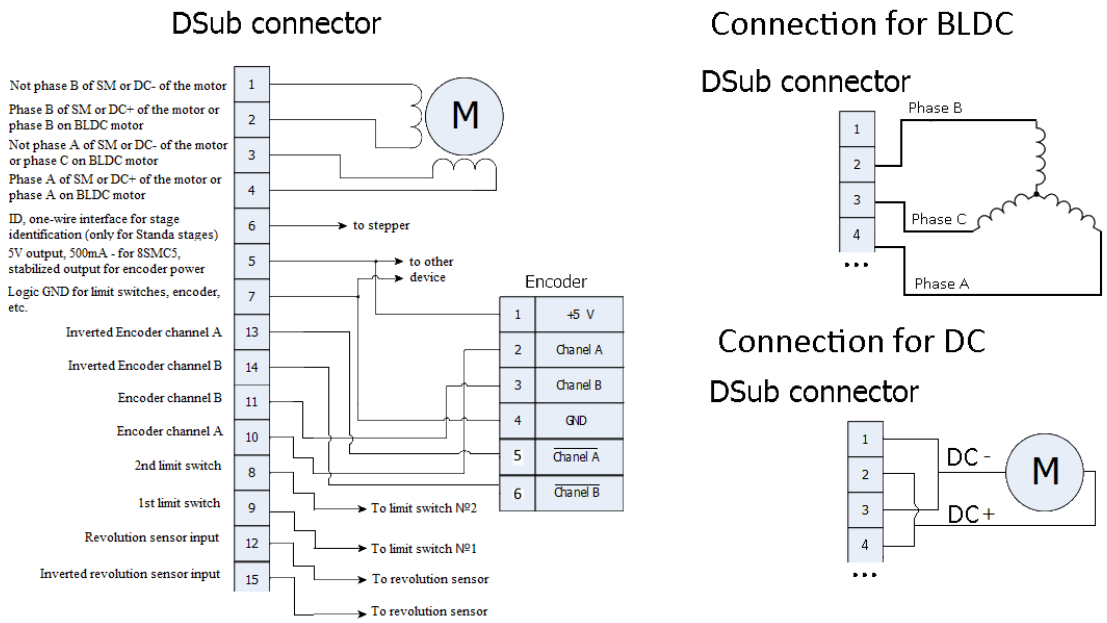


Fig. 3.12: General diagram of stage and encoder connection using D-sub connector

Note: If A and B encoder channels work in open drain mode, some extra pull-up of encoder outputs to 5V power voltage using the resistors may be required at high rotation speeds in order to provide the maximum signal transmission speed (see *Operation with encoders*).

Note: Only firmwares 4.1.0 and older support BLDC.

3.3.2 Example

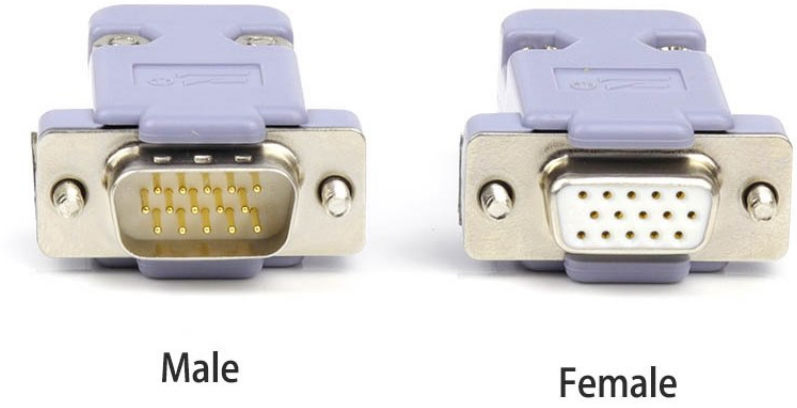
Consider the connection of the two-phase stepper motor Nanotec ST5918L3008-B with encoder CUI INC AMT112S-V to controller 8SMC5-USB.

3.3.2.1 Preparation

To get started, we need:

- Motor;
- Encoder;
- *Pinout of D-SUB connector* for 8SMC5-USB;

- Motor datasheet ;
- Encoder datasheet ;
- Soldering equipment: soldering-iron, wires, flux, solder, nippers, heat shrink tubes of different sizes;
- Screws M2.5x6 for fixing the encoder;
- D-SUB cover + connector (male) and wires for cable manufacturing;







3.3.2.2 Connecting the motor and encoder to the controller

- Before you begin, assemble the encoder in accordance with the appropriate instructions.

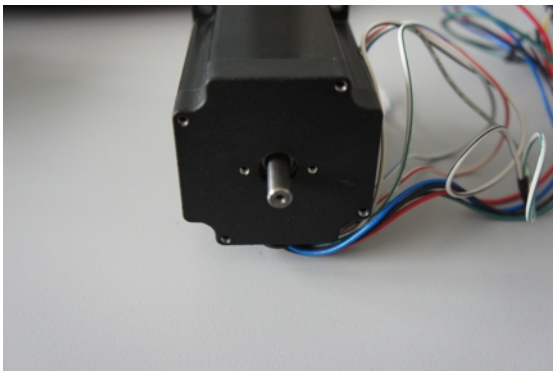


Fig. 3.13: The motor without encoder. Note 2 holes M2.5 to which is usually attached an encoder

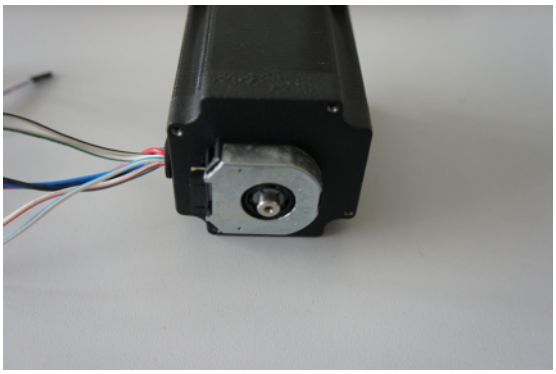


Fig. 3.14: Motor with attached encoder

- Let us look in the specification of the engine and find the wiring diagram (for Nanotec ST5918L3008-B it is at the bottom right in the specification):

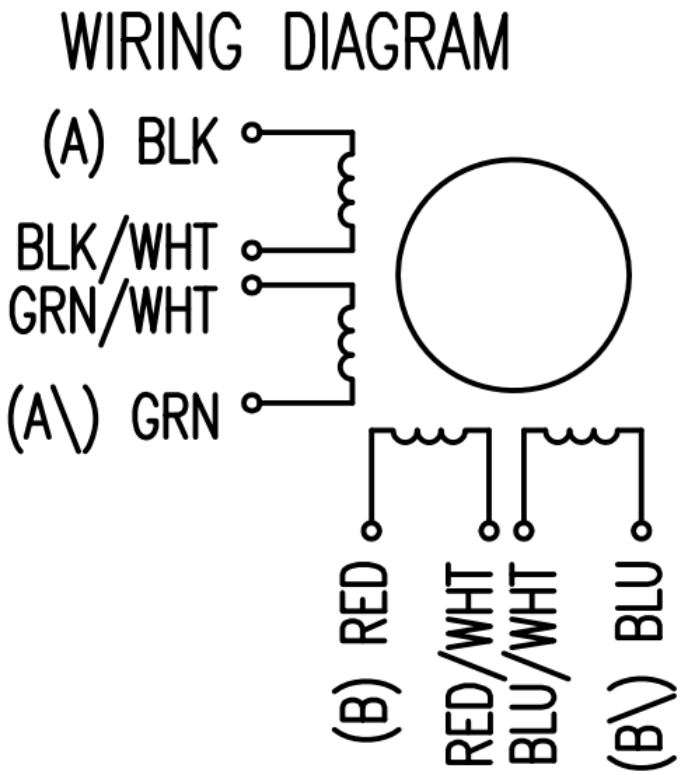


Fig. 3.15: Motor contacts

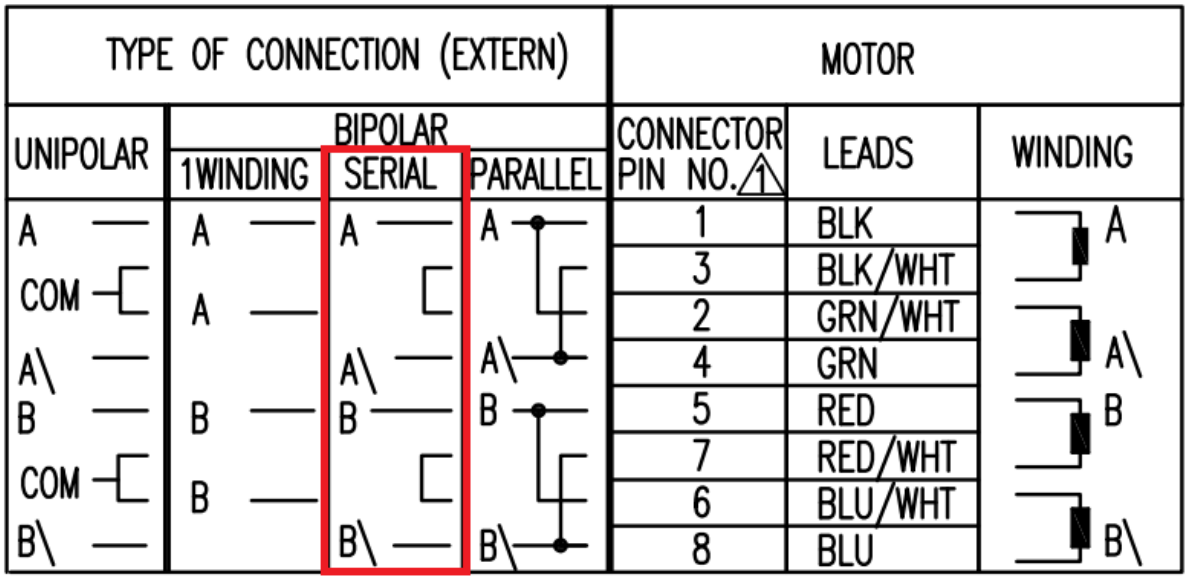
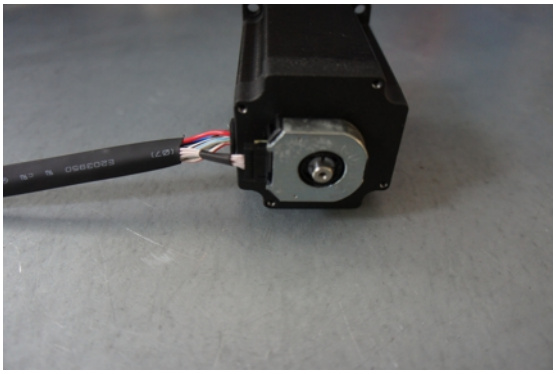


Fig. 3.16: Connection type

– There exist serial and parallel winding connection and each type allows to obtain various characteristics for the motor. We will connect the windings in series (red frame on the picture). To do this, wires having two colors BLK/WHT and GRN/WHT, as well as RED/WHT and BLU/WHT must be connected to each

other in pairs. Next, you need to put in accordance *A*, *not A*, *B*, *not B* pins of controller to contacts of motor windings ST5918L3008-B: black, green, red, blue. One winding is a connection of *A* and *not A* or *B* and *not B*. After the connection between a two-color wire, you will get that one winding of the motor is black - green connection, other is red - blue. Therefore, matching contacts will be the follows: black - *A*, green - *not A*, red - *B*, blue - *not B*. It can be seen in the picture above *Connection type*.

– To connect encoder, open its datasheet and find 5 contacts on encoder connector: *A+* (channel *A*), *B+* (channel *B*, shifted relative to *A* by 90 degrees), *Z+* (rev counter), *5V*, *GND*. They should be taken from the encoder as 5 separate wires and put together with the wires from the motor as they then go to a connector. CUI INC AMT112S-V encoder has 18 pin input, therefore it is needed to make a cable with the same connector on the end to output necessary signals:



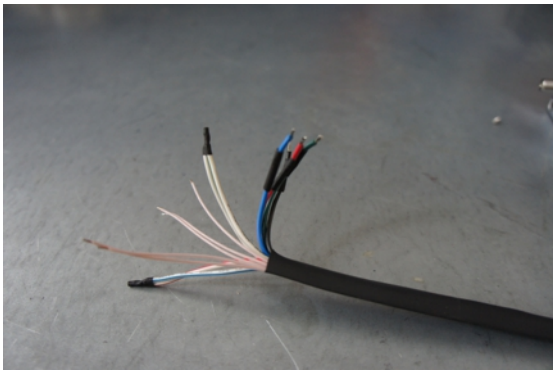
Encoder contacts A+, B+, Z+, 5V and GND corresponds to 10, 11, 12, 5, 7 pins of *D-SUB male connector* respectively.

For convenience, use the next tables (the number in parentheses indicates pin on the corresponding connector):

Encoder pin	**D-SUB pin**
A+ (10)	Encoder A (10)
B+ (8)	Encoder B (11)
Z+ (12)	Revolution sensor input (12)
5V (6)	Output 5V, 100 mA (5)
GND (4)	Logical ground (7)

Motor pin	**D-SUB pin**
A (BLK)	phase *A* (4)
not A (GRN)	phase *not A* (3)
B (RED)	phase *B* (2)
not B (BLU)	phase *not B* (1)

- Solder the above contacts to D-SUB male connector:



The wires from the motor and encoder in a heat shrink tube. Note the presence of small heat-shrinkable tubes for wires going to the motor windings (BLK, GRN, RED and BLU), as well as two-colored wires joined together (BLK/WHT and GRN/WHT, RED/WHT and BLU/WHT). The thin wires are an encoder contacts (5 pcs).

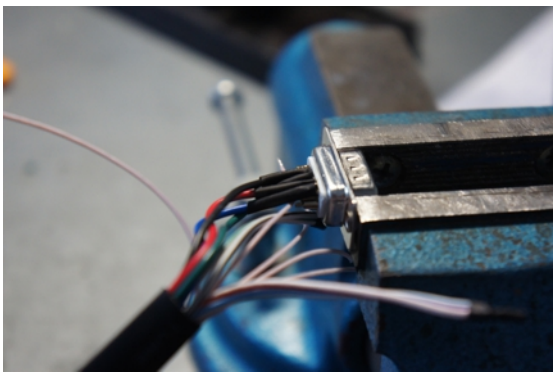


Fig. 3.17: Soldered contacts of the motor windings

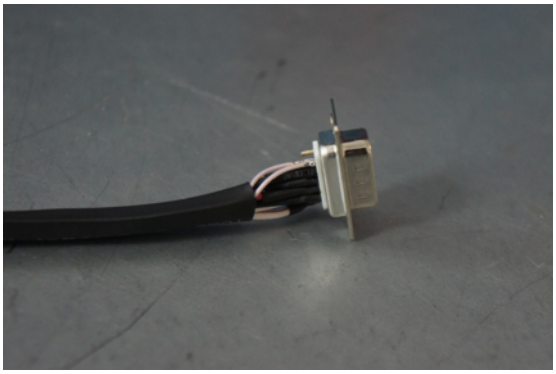
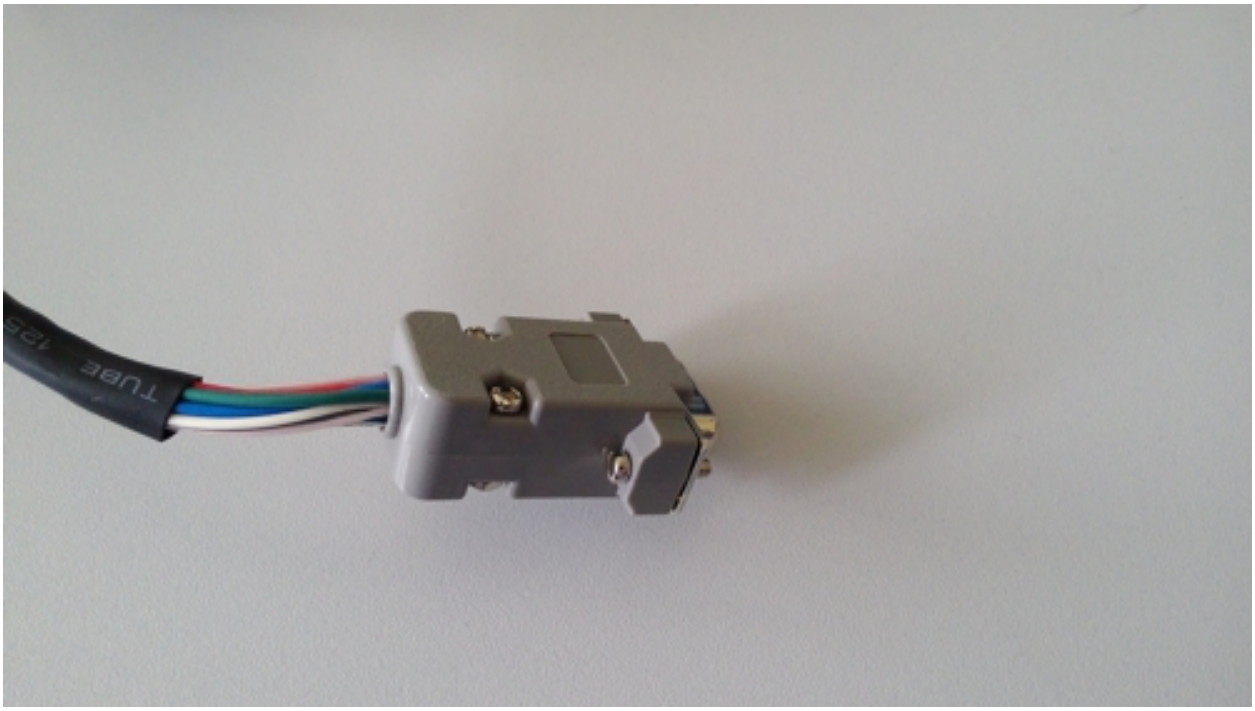


Fig. 3.18: Ready cable from the motor with the D-SUB connector on its end

Recommendation: use heat shrink tubes of a small diameter (2-3 mm) while soldering contacts to D-SUB connector, and large diameter to skip through them all the wires coming from the motor and encoder. Put them before soldering.

- Put D-SUB connector into the cover



- Now you can connect it to 8SMC5-USB.

Description and profile settings are given in the next chapter *Manual profile setting*.

3.4 Manual profile setting

- *Introduction*
- *Getting started*
- *Nominal current setting*
- *Basic parameters setting*
- *Hardware limit switches setting. Homing.*
- *Encoder parameters setting*
- *Setting the kinematic characteristics of the controller*
- *Working with user units*

3.4.1 Introduction

All necessary parameters are set after motor connection (see *Example of a motor connection* where the Nanotec ST5918L3008-B motor connection is described). There we will consider the setting of the profile for **Nanotec ST5918L3008-B** stepper motor.

3.4.2 Getting started

- Install and run XILab (see *Overview and getting started*).

- Load the profile with default settings. To do this, open **Settings -> Load setting from file...** and select xilabdefault.cfg file from XILab folder.

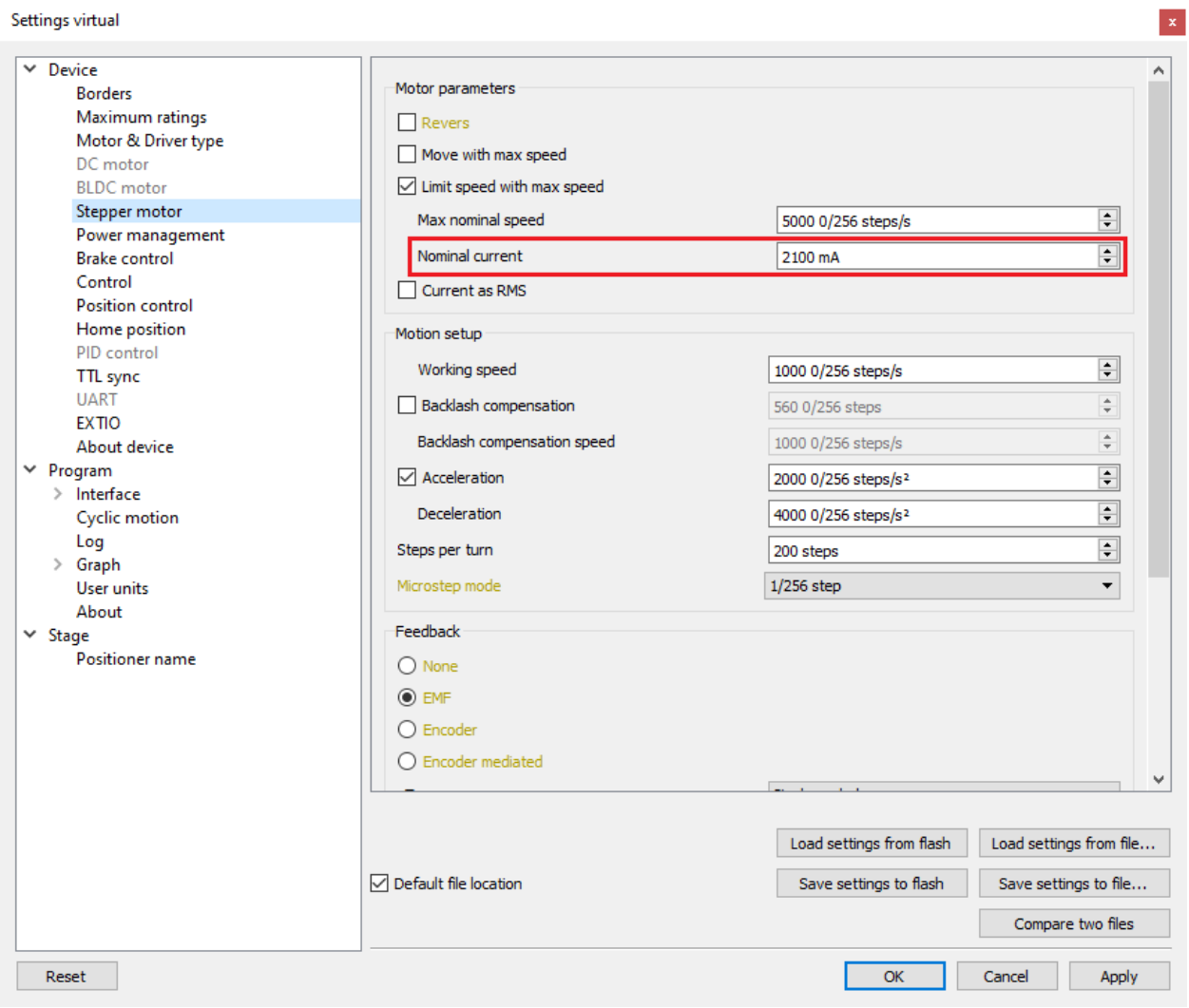
3.4.3 Nominal current setting

Initially, it is needed to set a correct current in motor windings:

- From the specification find the **phase current 2.1 A** - this is the maximum current for the motor in case of **serial** winding connection:

SPECIFICATION \ CONNECTION	UNIPOLAR OR BIPOLAR–1 WINDING	BIPOLAR	
		SERIAL	PARALLEL
VOLTAGE (VDC)	3.0		
AMPS/PHASE	3.0	2.1	4.2
RESISTANCE/PHASE (Ohms)@25°C	1.0±10%	2.0±10%	0.5±10%
INDUCTANCE/PHASE (mH) @1KHz	2.2±20%	8.8±20%	2.2±20%

- Being in the *Settings* window, open *Stepper motor* tab. There are such parameters as rotational speed, acceleration, driving mode, etc. (see *Settings of kinematics (stepper motor)* for additional information). In *Motor parameters-> Nominal current* field you should specify the value of the phase current **not exceeding 2.1 A**:



3.4.4 Basic parameters setting

- In the *Working speed* field we will specify a rotation speed. The recommended speed is not more than **1000 s/sec** at the first start. In the same window you should type *Max Nominal Speed* (**5000 s/sec** is reasonable value for majority of motors and motorized stages) and mark *Limit speed with max speed*. This setting is necessary to limit the motor speed since some mechanical systems can be designed for low speed, and fast rotation can lead to severe wear of motor/stage mechanics.
- In the motor specification we find the number of steps per rotation. For our motor this value is equal to **200 steps**. Specify it in the *Steps per turn* field. Usually, the value of one pitch in degrees is listed in motor description, on the basis of which you can calculate the number of steps per revolution, knowing that one revolution consists of 360 degrees.
- Make sure that movement to the right from the main window of XILab corresponds to the movement to the right of the stage. If not, then check the box *Reverse* field *Stepper motor -> Motor parameters*.

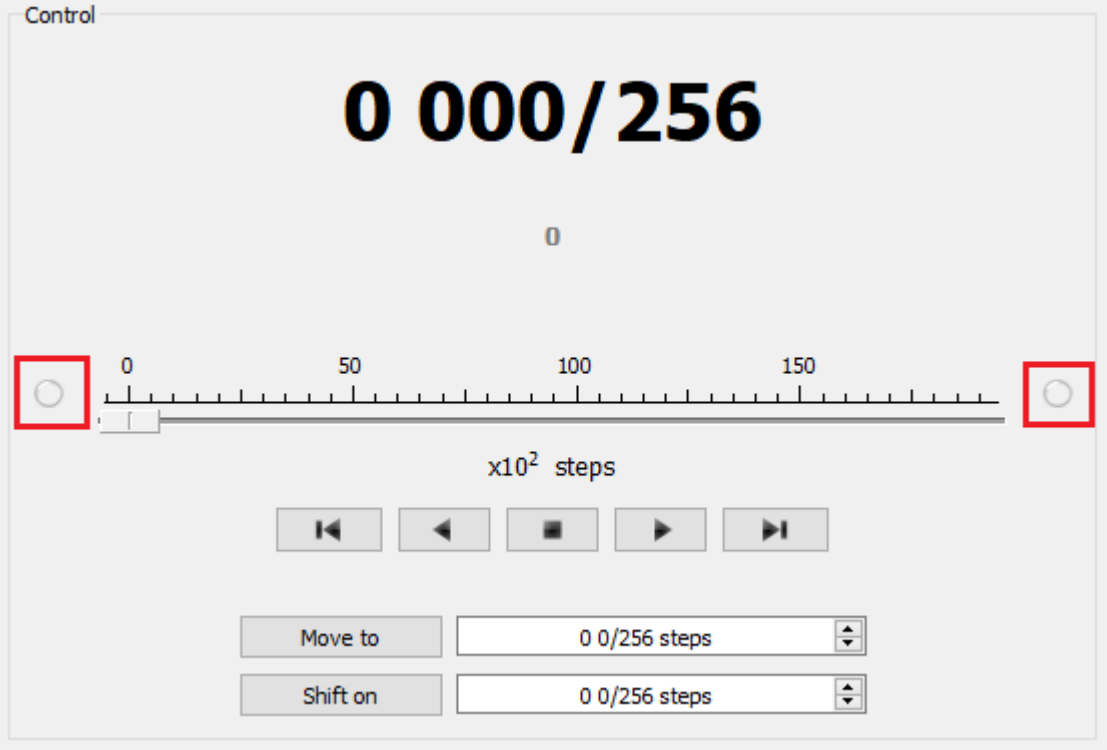
3.4.5 Hardware limit switches setting. Homing.

Note: This section describes the using of motorized stages with hardware *limit switches*. If your system is not provided with hardware limit switches, it is recommended to disable stop by limit switches in settings. To do this,

unmark *Stop at right border* and *Stop at left border* in *Borders* tab.

There are stages with limited (translators) and an unlimited range of motion (rotators). The limitation of movement range can be done by position or with limit switches using. When you work with translators if its limit switches are configured incorrectly there exists a risk to break down mechanics, since moving part can try to go out of motion range. Rotators do not have such problem. Moreover, it should be kept in mind that rotator may have only one limit switch.

- To work with limit switches you must specify which one will be left and right. Sometimes it is unknown in advance and we only know that both switches are connected and fire if the corresponding limit of the motion range is reached. The stage jam is possible if the limit switches are configured improperly. Therefore, the controller supports just a simple detection of incorrectly configured limit switches, shutting down the movement on both of them. Please make sure that:
  - The stage is far from limit switches;
  - Switches polarity is configured correctly (limit switches indicators are off in the main window of XILab). In the case of incorrect settings, change their polarity (*Borders -> Pushed position*), indicators should go out.



  - Shutdown mode is activated on both of limit switches (*Stop at right border* and *Stop at left border* are marked in *Borders* tab).
- Mark the flag detecting improper connection of limit switches *Border misset detection* in *Borders* tab.

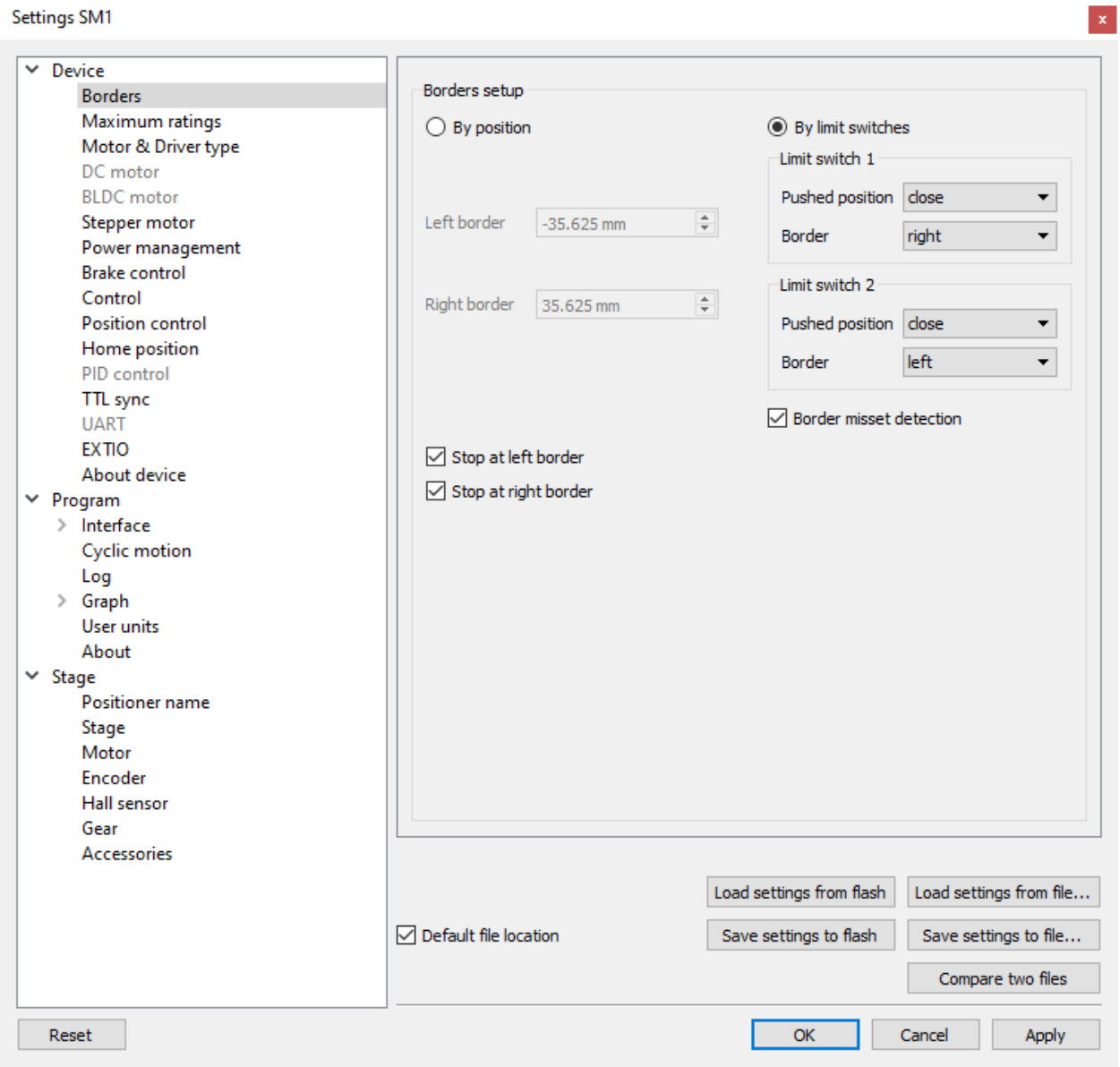


Fig. 3.19: Tab with limit switches settings

- Controller can switch to Alarm state after false limit switch response, if *Enter Alarm state when edge misset is detected* is enabled in *Maximum ratings* tab. It is recommended to enable it. Start the movement in any direction from the *main window of XILab* up to Alarm state or stopping by the limit switch. When an Alarm occurs you need to reverse limit switches by changing *Borders->Border* with reversed values in the *Stepper motor* tab.

Warning: The protection against mistaken limit switches connection doesn't guarantee the complete solution of the problem, it only makes the initial configuration procedure easier. Don't start the movement with mixed up limit switches if any of them is active, even if the protection is on.

There are still two ways to determine which of the limit switch is right and which is the left:

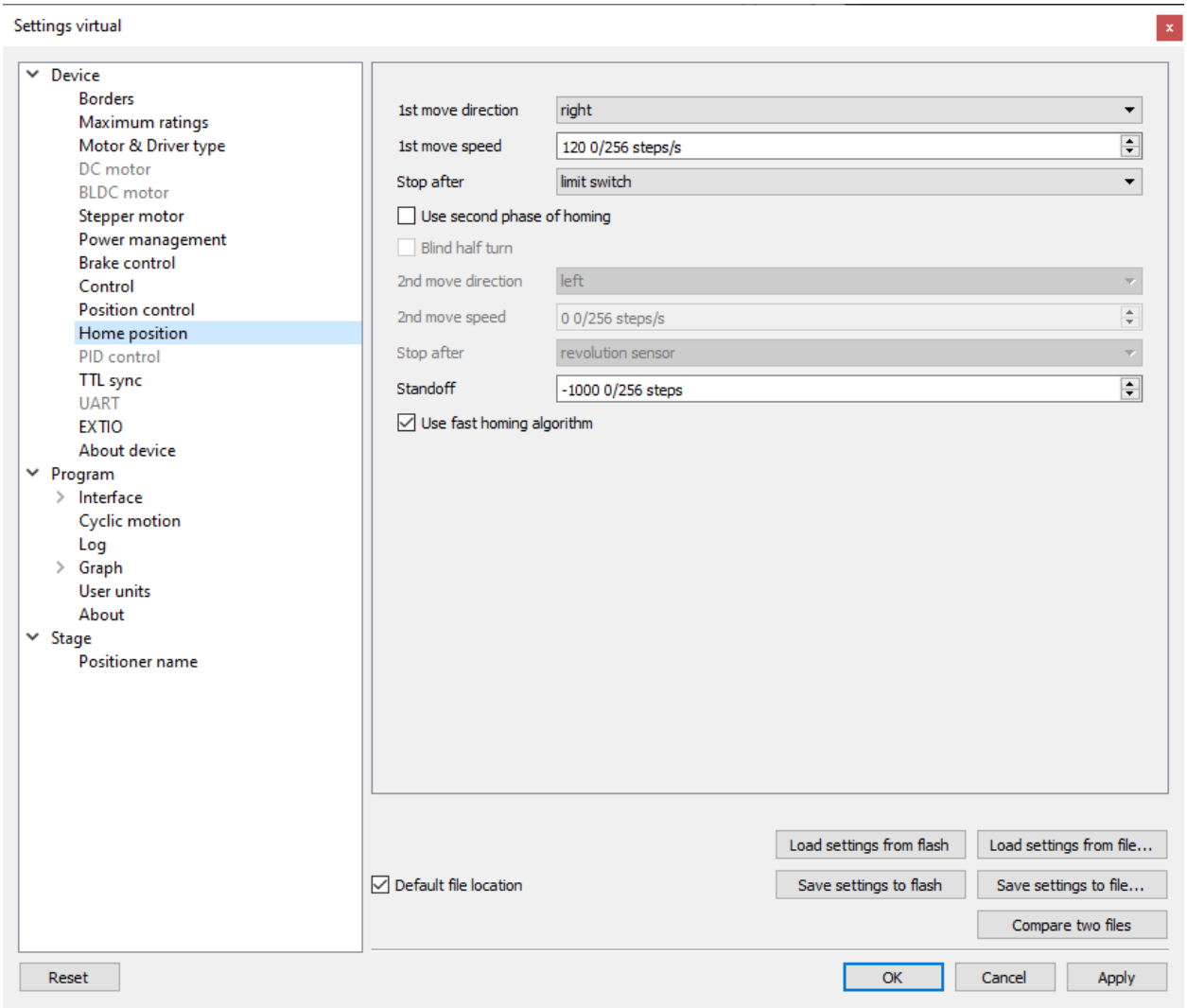
- You need to know how each of the limit switches is connected to the stage. When loading a profile with the

default settings, switch connected to pin 9 of the D-SUB connector is considered as left, while switch connected to pin 8 - as right. Their location relative to the stage is configured in the fields *Limit switch 1* and *Limit switch 2* (see screenshot above). Start the system at the low speed (<100 steps/s) when it is far away from limit switches. If the direction of movement to the switch in a real setting differs from the expected, change *Borders->Border* values with reversed in the *Stepper motor* tab.

- If it is possible to get limit switches activate them and note the correspondence between indicators in XILab and each particular switch. Then start the system at the low speed (<100 steps/s) when it is far away from limit switches and make sure that the system moves to the right switch. Compare this to what you see in the main window of XILab. If the direction of movement to the switch in a real setting and in the main window differs, change *Borders->Border* values with reversed in the *Stepper motor* tab.

For detailed information refer to *motion range and limit switches*.

- Controller has a useful function called *automatic Home position calibration* to set the initial position of the motorized stage.



We will consider the most simple configurations with a single phase only. Start from the setting of the *1st phase speed* which is approximately 5-10 times lower than *Working speed*. It is necessary for higher precision of automatic calibration procedure. In the filed *Stop after* specify the *limit switch* to make the stage reached one of the limit switches

during the calibration (direction is selected in the *1st move direction*). In the field *Standoff* specify number in steps, for which stage must be driven away from the limit switch. Click *Ok* or *Apply*.

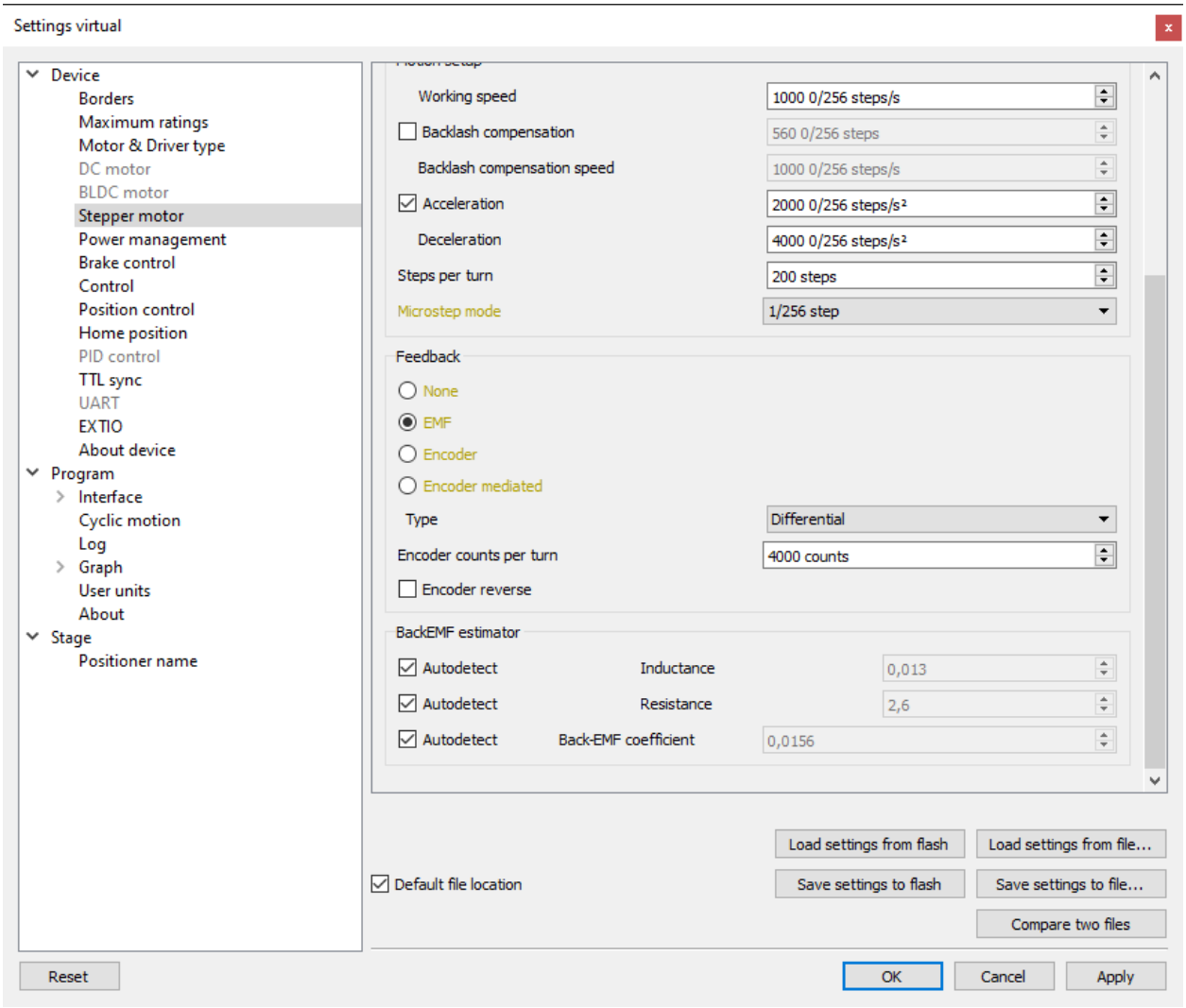
Note: *Standoff* value is signed. Positive direction is right. That is, if the auto-calibration procedure is set up on the right limit switch, then in order to move stage away to the left you should type negative value in *Standoff* field.

- Start the automatic calibration by clicking *Go home* in the main window of XILab. The result of it is a movement of the stage to the specified limit switch with a relatively low speed and the shift away from him to the value specified in the field *Standoff*.
- After completion of the calibration process, press *ZERO* in XILab to set the origin of coordinate system for your stage.
- Repeat the calibration process again. The stage must return to the ZERO position. Please pay attention that there can be slight deviations from ZERO connected with calibration procedure error.

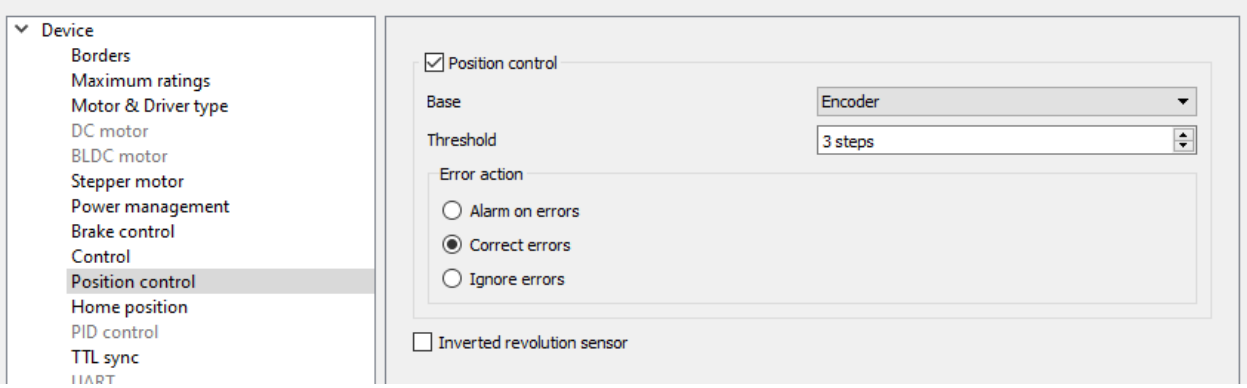
3.4.6 Encoder parameters setting

Note: This section describes the using of motor with encoder. If you motor without an encoder, the parameters described below can be left unchanged.

- Any encoder has **Pulse Per Turn** - **PPT** parameter (sometimes it is called **PPR** - **Pulse Per Rotation**). For correct operation of the encoder with controller you should enter the number of encoder counts per revolution, which is equal to **4xPPT** in the *Encoder counts per turn* field in XILab. For example, if your encoder has **1024** pulses per turn, specify **4096** in the *Counts per turn*:

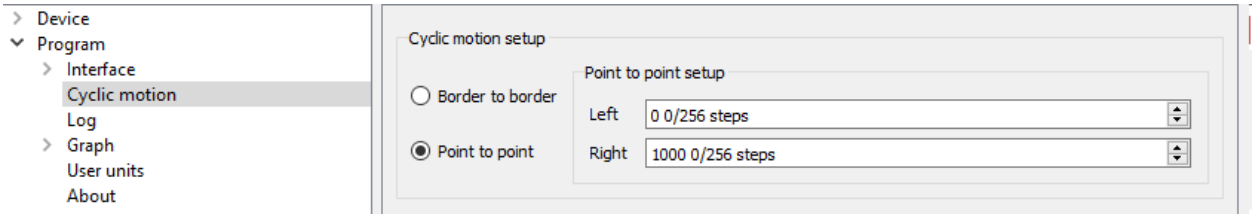


- Start the motor rotation from the *main window of XILab*. If everything is configured correctly, the green indicator ENCD will light in the bottom of window. If ENCD has yellow color, you should mark *Encoder reverse* in the *Stepper motor* tab. Red color of EDCN points to the problem with encoder position recalculation.
- It is possible to activate the position control by encoder. To do this, in the tab *Position control* mark *Position control* and specify allowable error in terms of encoder counts in the *Threshold* field. Then, when a mismatch between position and encoder counts occurs, indicator SLIP will light in the bottom of XILab main window. Beyond this, if *Alarm on errors* is marked, the controller will go to Alarm state. *Correct errors* allows you to start closed loop control, when the difference between real position and encoder position is compensated.



3.4.7 Setting the kinematic characteristics of the controller

- In the *Stepper motor* tab you may specify a necessary acceleration (*Acceleration*) and deceleration (*Deceleration*) for your stepper motor. The process of optimal values selection is the next:
  - Starting from default values make small shifts (start and fast stop) with gradual *Acceleration* increase until the movement become unstable and disrupted sometimes. Take acceleration equal to about **half** of this value.
  - The deceleration can be configured about **1.5 - 2 times higher** than acceleration.
- If in your mechanical system moving to the desired position on the left and on the right is not the same, and there is play, it is possible to eliminate this ambiguity. To do this mark *Backlash compensation* in *Stepper motor* and type number greater than play value. The sign of this setting determines the direction of moving to the position. Positive sign means move from the left while negative - from the right. In *Backlash compensation speed* field set the speed of compensation movement. This value should be low (**50 s/sec** is enough) in order to avoid "drifting" during backlash compensation.
- After the basic configuration of the stage/motor, you can increase working speed. It can be done experimentally like the process of acceleration setting, i.e. you should take its value **about 2 times lower** than value at which there is unstable movement. To test the stability of the rotation it is recommended to use the function *Cyclic* in XILab *main window*. Make sure that you *set* it previously.



- In the *Microstep mode* field we recommend to enter the value **1/256**.

3.4.8 Working with user units

Often it is uncomfortable to work with the steps and microsteps and more convenient units are preferable. For this reason, the controller can recalculate the coordinates in the usual units, for example in millimeters or degrees. It can be done in the tab *User units*, where you should specify the size of the step and the corresponding measurement unit. For more information, refer to *relevant documentation paragraph*.

Configuration of the operating profile complete.

3.5 Calculation of the nominal current

In order to stepper engine gave maximum torque, but it does not overheat, it is important to specify such technical characteristic as the rated current.

The greater a current in the motor winding, the greater the torque at the axis. It is important to remember that with an increase a current flowing through the winding, thermal power released by the motor increases. So the engine could operate for a long time allocated to thermal power (Joule heating) must be less power dissipation. Power dissipation can be calculated on the basis of documentation on the engine.

3.5.1 Calculation based on the parameters of unipolar full step mode

Power dissipation is equal to

$$P = n \cdot R_u I_u^2,$$

where R_u - the resistance of the winding in unipolar mode, I_u - current through the winding in unipolar mode, n - the number of simultaneous windings.

Consider, for example, ST2818M1006. The table in the documentation shows that in full step mode simultaneously running two phase (n = 2) in the unipolar mode, i.e. $P = 2R_u I_u^2$. The motor controllers support only bipolar control mode. To switch from a unipolar to a bipolar mode, connect each phase windings in series, the resistance will increase, $R_b = 2R_u$, where R_b - the resistance of the series-connected windings in the bipolar control mode.

The motor controllers control algorithm is capable of operating in a microstepping mode and maintains the current so that the first winding current varies in function $I_a \sin(\phi)$, in the other winding current varies in function $I_a \cos(\phi)$, where I_a - current amplitude. Thermal power released two windings at any time

$$P = R_b I_a^2 \sin^2(\phi) + R_b I_a^2 \cos^2(\phi) = R_b I_a^2$$

It follows from the foregoing that the $I_a = I_u$.

3.5.2 Calculation based on the parameters of bipolar full step mode

Power dissipation is equal to $P = n \cdot R_b I_b^2$, where R_b - the resistance of the winding in bipolar mode, I_u - current through the winding in bipolar mode, n - the number of simultaneous windings.

Consider, for example, ST2018S0604. The table in the documentation shows that in full step mode simultaneously running two phase (n = 2) in the bipolar mode, i.e. $P = 2R_b I_b^2$.

Thermal power dissipated in the motor windings that managed by motor controller, still is

$$P = R_b I_a^2 \sin^2(\phi) + R_b I_a^2 \cos^2(\phi) = R_b I_a^2$$

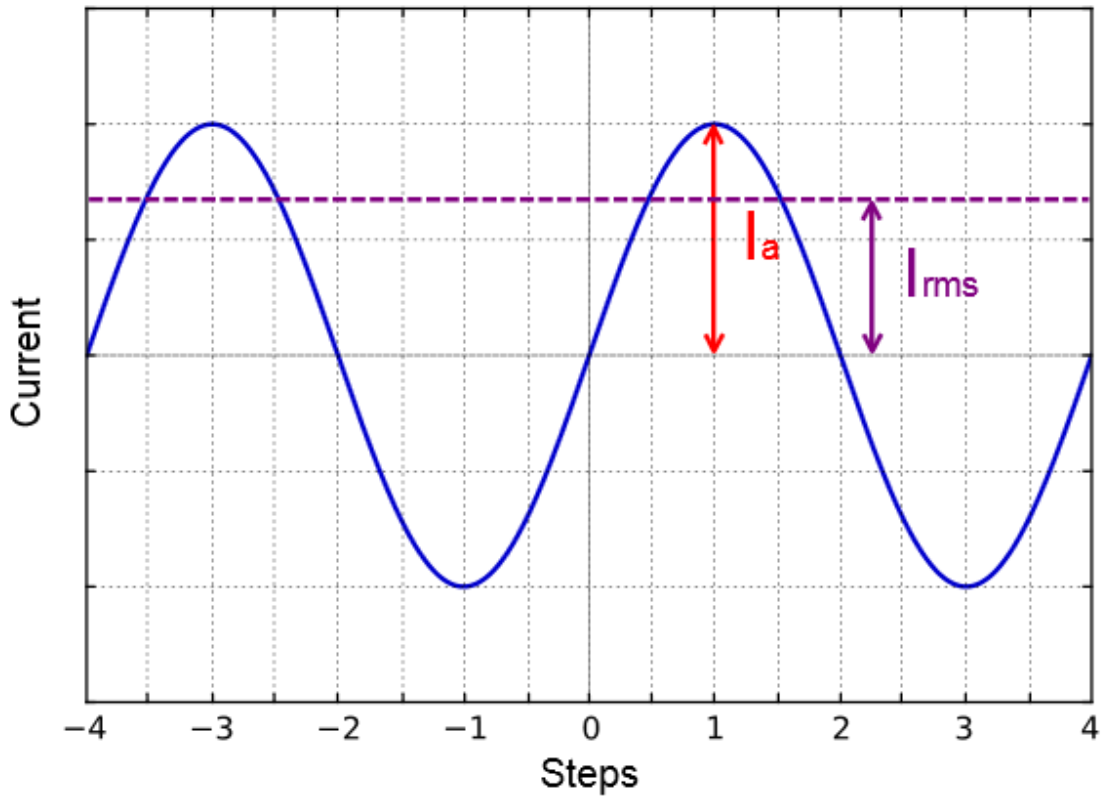
We obtain the equation equating power $2R_b I_b^2 = R_b I_a^2$. We find that $I_a = \sqrt{2} \cdot I_b$.

3.5.3 The relationship with an rms current

Alternating current in each motor winding can be characterized by its rms value in the period

$$I_{rms} = \sqrt{\frac{1}{2\pi} \int_0^{2\pi} (I_a \sin(\phi))^2 \, \mathrm{d}\phi} = \frac{I_a}{\sqrt{2}}$$

Thermal power of **one** winding is associated with an rms current through it $P_1 = R_b I_{rms}^2$. Both windings are identical $P_1 = P_2$. The total thermal power of the engine that is run by control by motor controller controller $P = P_1 + P_2 = 2R_b I_{rms}^2$.



It follows from the foregoing that $I_{rms} = \frac{I_u}{\sqrt{2}}$, also $I_{rms} = I_b$.

3.5.4 Amplitude and rated current for BLDC

The rated motor current is calculated from the maximum allowable heat dissipation. The rated current written in the documentation is calculated from the power limit allocated when the power supply is connected to the two windings.

Let's write the formula for power with this connection:

$$P_{chop} = 2R_{phase}I_{rate}^2$$

Formula for the power generated by the windings for sinusoidal control:

$$P_{sin} = 3R_{phase}I_{rms}^2$$

The rated motor current is calculated from the maximum allowable heat dissipation. Equate right parts of formulas:

$$I_{rms} = \frac{\sqrt{2}}{\sqrt{3}}I_{rate}$$

So,

$$I_{amp} = \frac{2I_{rate}}{\sqrt{3}}$$

This means that if the documentation on your engine says that the rated current is, for example, 0.88A, then a amplitude current value can be written to the controller:

$$I_{amp} = \frac{2 * 0.88}{\sqrt{3}} = 1A$$

3.5.5 Setting the nominal current

Motor controller are capable of taking the nominal current value as a current amplitude or as rms. The choice of which way to interpret the input value of the nominal current is determined by the absence or presence corresponding flag *ENGINE_CURRENT_AS_RMS* in the *EngineFlags* *engine settings* structure. When *setting the nominal current in XILab* should properly specify how the current is interpreted. Motor controllers in this case will provide the maximum torque without overheating the engine.

The same flag also controls the semantics of the BLDC current.

As for the stepper, there is a special checkbox in the XiLab for BLDC, which determines how to interpret the value entered in the Nominal current field. If checkbox “Amplitude current” is checked, the entered current value will be amplitude: maximum the amplitude of the sine will always be less than this value. If checkbox “Amplitude current” is cleared, the verified value will be recalculated by the formula for and the current amplitude will be limited already by this recalculated value

For all Standa motorized stages prepared configuration files that contain the specified nominal current as rms. The corresponding flag is set. Thus the engines operate at optimum settings.

CHAPTER

FOUR

TECHNICAL SPECIFICATION

4.1 Appearance and connectors

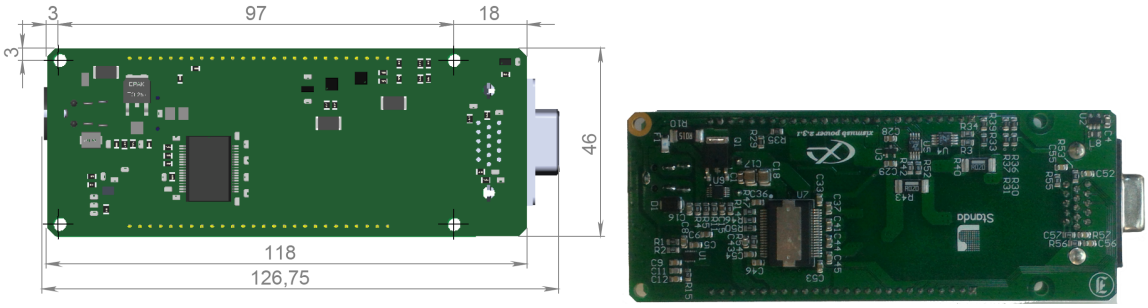
8SMC5-USB controllers are available in 3 different versions:

- *Controller board*
- *One axis system*
- *Two axes system*

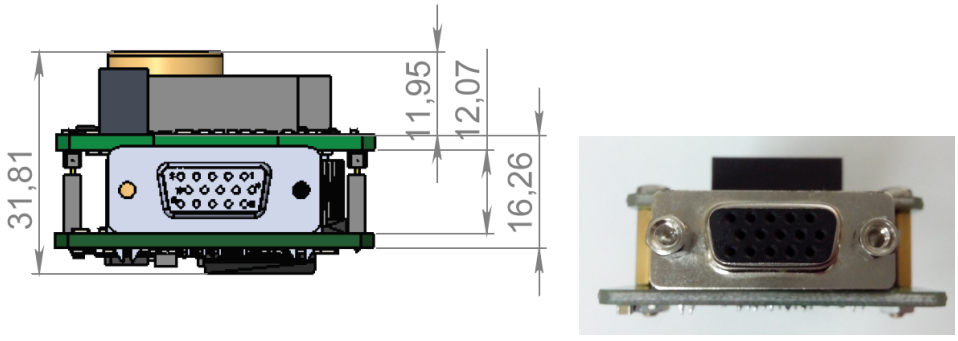
4.1.1 Controller board

4.1.1.1 Dimensions and arrangement

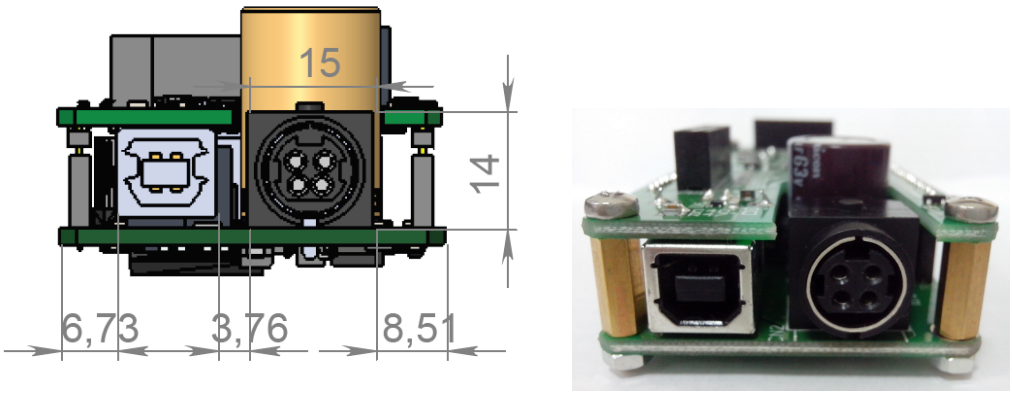
Structurally the controlled is designed as two boards, 127 x 46 x 32 mm each, rigidly connected to each other. A power part mounted on the bottom board, the logic controller and control systems are located on the top board. A radiator at the power part is available.



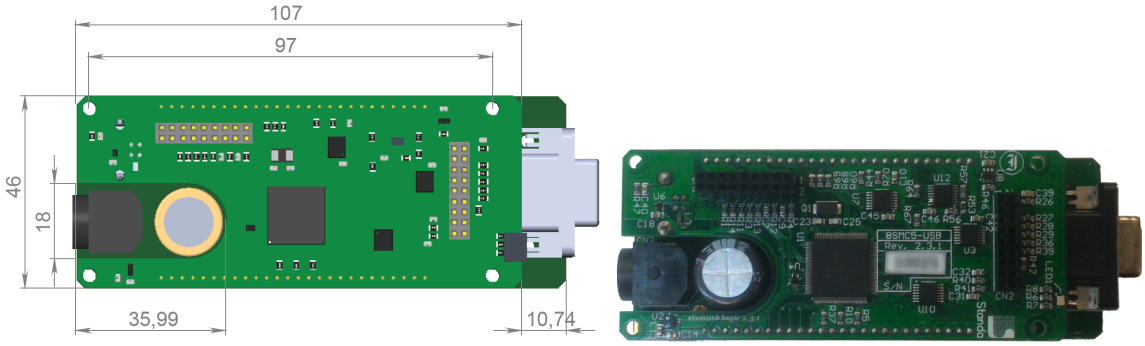
Top view on the controller. The view from power part and radiator side.



Front view on the controller. The view from stage cable side.



Rear view on the controller. The view from DIN and USB type-B connectors side.



Bottom view on the controller. The view from backplane connectors side.

Important: If you are mounting the radiator to the power part by yourself, please make sure that there is no contact between heat-conducting surfaces and conductive elements of the unit. Such contact may damage the power circuit! This warning is applicable only to controllers supplied without body.

4.1.1.2 Controller board connectors

4.1.1.2.1 Stage connector

A female DSub 15-pin connector for stage is mounted on the controller board.

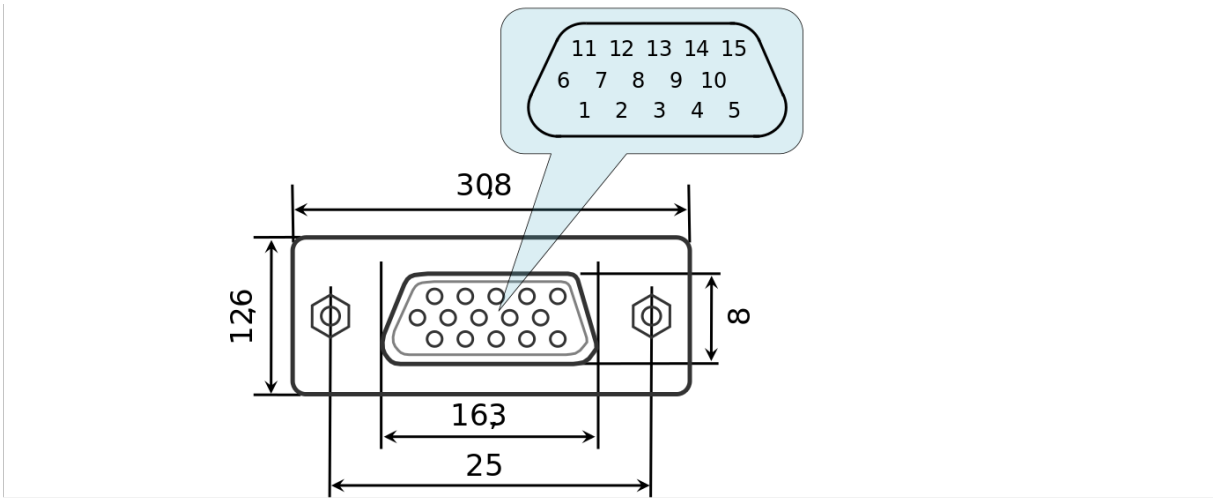


Fig. 4.1: Dimensions and numbers of the pins in DSub connector (front view)

Pins functionality:

1. Not phase B of SM or DC- of the motor
2. Phase B of SM or DC+ of the motor or phase B on BLDC motor
3. Not phase A of SM or DC- of the motor or phase C on BLDC motor
4. Phase A of SM or DC+ of the motor or phase A on BLDC motor
5. 5V output, 500mA - for 8SMC5, stabilized output for encoder power supply
6. ID, one-wire interface for stage identification (only for Standa stages)
7. Logic GND for limit switches, encoder, etc.
8. 2nd limit switch
9. 1st limit switch
10. Encoder channel A
11. Encoder channel B
12. Revolution sensor input
13. Inverted Encoder channel A
14. Inverted Encoder channel B
15. Inverted revolution sensor input

Note: Only firmwares 4.1.0 and older support BLDC.

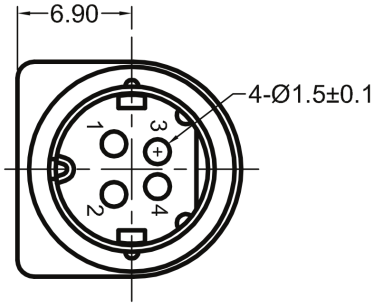
Note: Outputs 1 & 3 and 2 & 4 must be connected together for proper DC motor function if the nominal current of the motor is higher than 3A.

Warning: Plugging in/out the motor to the controller is not recommended while motor windings are under voltage.

4.1.1.2.2 Power supply connector

One and two-axis controller models use Kycon 4-pin DC power connector (part number KPPX-4P, www.kycon.com).





Pinout:

1. Power, "-".
2. Power, "+". 12-48V.
3. Power, "-".
4. Power, "+". 12-48V.

Important: Never supply the power to the controller and do not plug it to power connector if you are not confident that your power supply parameters conform to the requirements. Never attempt to plug the power supply to the controller if you are not sure power supply unit and controller connectors are compatible! The acceptable connection parameters are described in *Safety instructions*.

Important: Hot-swapping or unreliable connection of the power supply connector Kycon may damage the PC and/or the controller. For more details please refer to *Safety instructions*.

4.1.1.2.3 Data connector

Controllers connect via USB type-B connector.



Fig. 4.2: USB type-A - USB type-B cable

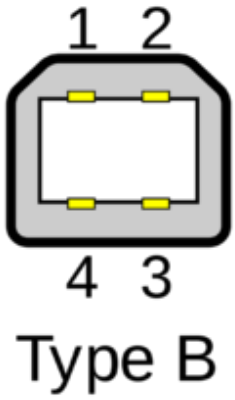


Fig. 4.3: USB type-B connector

Table 4.1: Output pin table

Pin #	Name	Wire colour	Description
1	VCC	Red	+5V DC
2	D-	White	Data -
3	D+	Green	Data +
4	GND	Black	Ground

Warning: Use verified USB cables only! Damaged or low-quality USB cable may cause improper controller operation, including motor rotation errors and errors of device recognition by PC operating system. Short cables with thick wires and screening are ideal for sustainable connection.

4.1.1.3 Backplane connector

A female 20-pin double-row connector (PBD-20R) with 2.54mm pitch is mounted on the controller board for backplane connection.

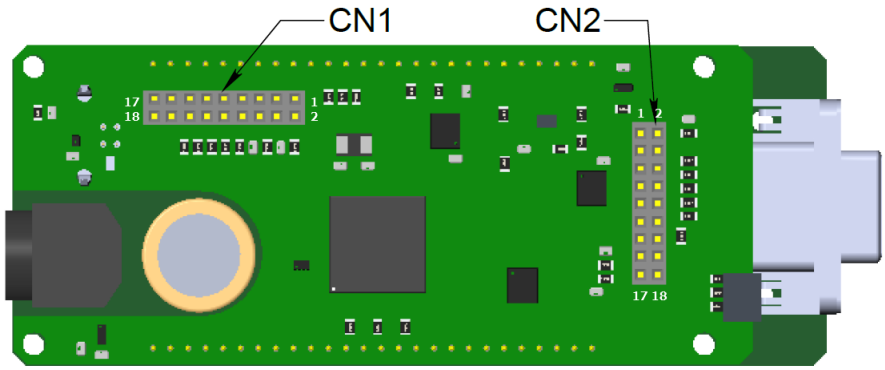


Fig. 4.4: Dimensions and output pins numbers in BPC (BackPlane Connector), connector-side view

Pin function of CN1:

1. Power GND 12-48V.
2. JOY, an analog 0-3 V input used for external *joystick* connection.
3. Power GND 12-48V.
4. POT, an analog 0-3 V input used for *general purpose*.
5. Power GND 12-48V.
6. EXTGPIO0, a common input-output (or an Enable signal if an external driver is used), 3.3V logic.
7. Power +12...+48V.
8. U_LED, a LED Status output displaying controller's operating mode.
9. Power +12...+48V.
10. L_LIM, a LED output for left limit switch.
11. Power +12...+48V.
12. R_LIM, a LED output for right limit switch.
13. USB_SEL, selecting USB device in Multi-axis system.
14. EXTGPIO1, a common input-output, 3.3V logic.
15. USB_D0_N, negative out of USB Slave.
16. H_USB_D0_N, negative out of USB Master.
17. USB_D0_P, positive out of USB Slave.
18. H_USB_D0_P, positive out of USB Master.

Pin function of CN2:

1. Reserved.
2. RX1, *serial port*, 3.3V logic.
3. Reserved.
4. TX1, *serial port*, 3.3V logic.
5. Reserved.

6. DIR, a direction signal for external driver control, 3.3V logic.
7. Reserved.
8. STEP, a step signal for external driver control, 3.3V logic.
9. Reserved.
10. SYNC_OUT, synchronization output, 3.3V logic.
11. Reserved.
12. SYNC_IN, synchronization input, 3.3V logic.
13. EMBRAKE, an output for magnetic brake control, 3.3V logic.
14. 3.3V logic power (relevant for logic boards version 2.3.6 and older).
15. BUT_R, an external *"Right" button*.
16. 5V output.
17. BUT_L, an external *"Left" button*.
18. Digital GND for 3.3V and 5V.

Note: Reserved contacts of the internal connector do not require any additional connection or tightening to the ground/power supply. **Just don't use them.**

Attention: A maximum designed operating voltage for Joy and Pot analog inputs is UP TO 3V. Never supply these inputs with higher voltage, including 3.3V value, otherwise the proper operation of all the analog channels of the controller may get disrupted and the controller itself or the motor may get damaged.

The outputs available at the internal connector are not protected. Mounting of any additional buffers or filtering chains **is a total responsibility of user or developer** who designs a backplane intended to use these lines.

While the controller is unpowered, no voltage over 0.3V relative to the DGND pin is allowed on the internal connector. Provision of any additional protection preventing those accidents (if possible) **is a total responsibility of user or developer** who designs a backplane intended to use these lines.

4.1.2 One axis system

Single-axis controller model is a *controller board* in a metal case. Case dimensions are 124 x 68 x 48,5 mm.

Front panel of the controller contains *power supply connector*, *USB type-B data connector*, status LED, power LED, left and right limit switch LEDs, left and right movement buttons.

Rear panel contains *stage connector*.

4.1.2.1 Connectors

4.1.2.1.1 Stage connector

A female DSub 15-pin connector for stage is mounted on the controller board.

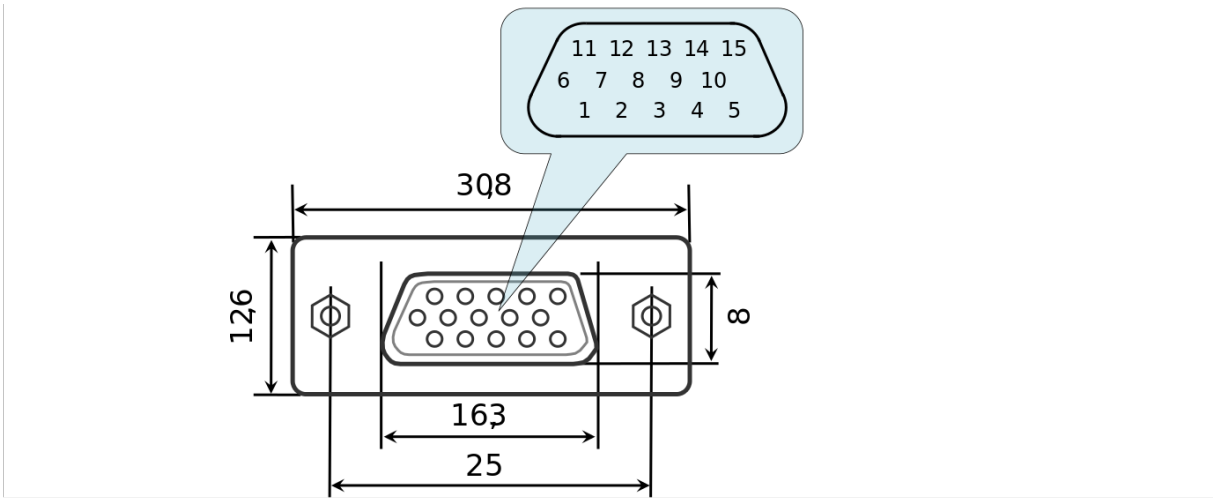


Fig. 4.5: Dimensions and numbers of the pins in DSub connector (front view)

Pins functionality:

1. Not phase B of SM or DC- of the motor
2. Phase B of SM or DC+ of the motor or phase B on BLDC motor
3. Not phase A of SM or DC- of the motor or phase C on BLDC motor
4. Phase A of SM or DC+ of the motor or phase A on BLDC motor
5. 5V output, 500mA - for 8SMC5, stabilized output for encoder power supply
6. ID, one-wire interface for stage identification (only for Standa stages)
7. Logic GND for limit switches, encoder, etc.
8. 2nd limit switch
9. 1st limit switch
10. Encoder channel A
11. Encoder channel B
12. Revolution sensor input
13. Inverted Encoder channel A
14. Inverted Encoder channel B
15. Inverted revolution sensor input

Note: Only firmwares 4.1.0 and older support BLDC.

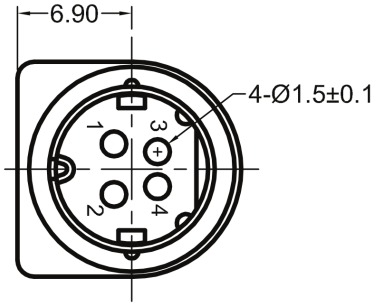
Note: Outputs 1 & 3 and 2 & 4 must be connected together for proper DC motor function if the nominal current of the motor is higher than 3A.

Warning: Plugging in/out the motor to the controller is not recommended while motor windings are under voltage.

4.1.2.1.2 Power supply connector

One and two-axis controller models use Kycon 4-pin DC power connector (part number KPPX-4P, www.kycon.com).





Pinout:

1. Power, "-".
2. Power, "+". 12-48V.
3. Power, "-".
4. Power, "+". 12-48V.

Important: Never supply the power to the controller and do not plug it to power connector if you are not confident that your power supply parameters conform to the requirements. Never attempt to plug the power supply to the controller if you are not sure power supply unit and controller connectors are compatible! The acceptable connection parameters are described in *Safety instructions*.

Important: Hot-swapping or unreliable connection of the power supply connector Kycon may damage the PC and/or the controller. For more details please refer to *Safety instructions*.

4.1.2.1.3 Data connector

Controllers connect via USB type-B connector.



Fig. 4.6: USB type-A - USB type-B cable

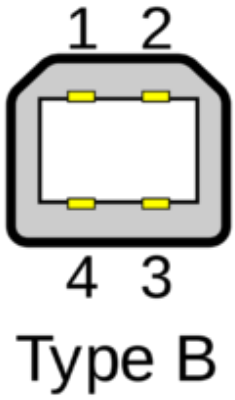


Fig. 4.7: USB type-B connector

Table 4.2: Output pin table

Pin #	Name	Wire colour	Description
1	VCC	Red	+5V DC
2	D-	White	Data -
3	D+	Green	Data +
4	GND	Black	Ground

Warning: Use verified USB cables only! Damaged or low-quality USB cable may cause improper controller operation, including motor rotation errors and errors of device recognition by PC operating system. Short cables with thick wires and screening are ideal for sustainable connection.

4.1.2.1.4 Supplementary one-axis system connector

One-axis controller model contains a HDB-26 female DSub connector.

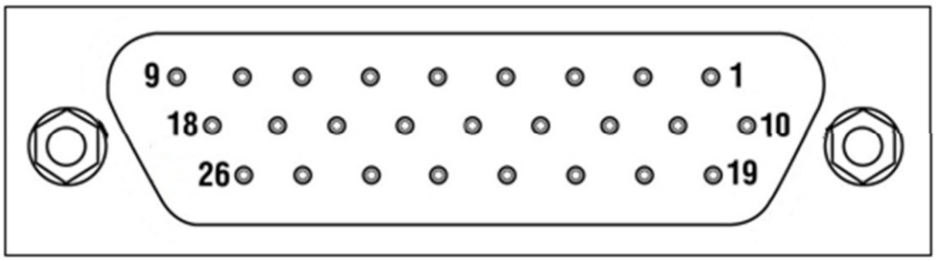


Fig. 4.8: Pinout of the supplementary HDB-26 connector, front view

Pinout:

1. NC, not used
2. NC, not used
3. NC, not used
4. DGND, digital ground
5. NC, not used
6. SYNC_IN, synchronization input
7. SYNC_OUT, synchronization output
8. RX, serial port input
9. NC, not used
10. NC, not used
11. NC, not used
12. NC, not used
13. DIR, direction signal for the external driver
14. DGND, digital ground
15. +5V
16. EMBRAKE, brake control output
17. CLOCK, clock signal for the external driver
18. TX, serial port output
19. NC, not used
20. NC, not used
21. NC, not used
22. PGND, power ground
23. PBRK (24V), magnetic brake output
24. +5V
25. EXTGPIO_0, *input-output pin*
26. POT, *analog input*

4.1.3 Two axes system

4.1.3.1 Enclosure view

Two-axis controller model consists of two *controller boards* in a metal case. Case dimensions are 124 x 45 x 106 mm.

Front panel contains *power supply connector*, *USB type-B data connector*, *supplementary two-axis system connector*, USB cascade connector, and power LED. The cascade USB type-A output connector is used to connect several two-axis cases in line terminated with either one-axis or two-axis case. This way a required number of axes can be connected to the computer with a single USB cable. The front panel also contains status LED, left and right limit switch LED, left and right movement buttons for each of the two controller boards of the two-axis system. Rear panel contains *stage connector* for each of the two controller boards, a *joystick connector*.

4.1.3.2 Connectors

4.1.3.2.1 Stage connector

A female DSub 15-pin connector for stage is mounted on the controller board.

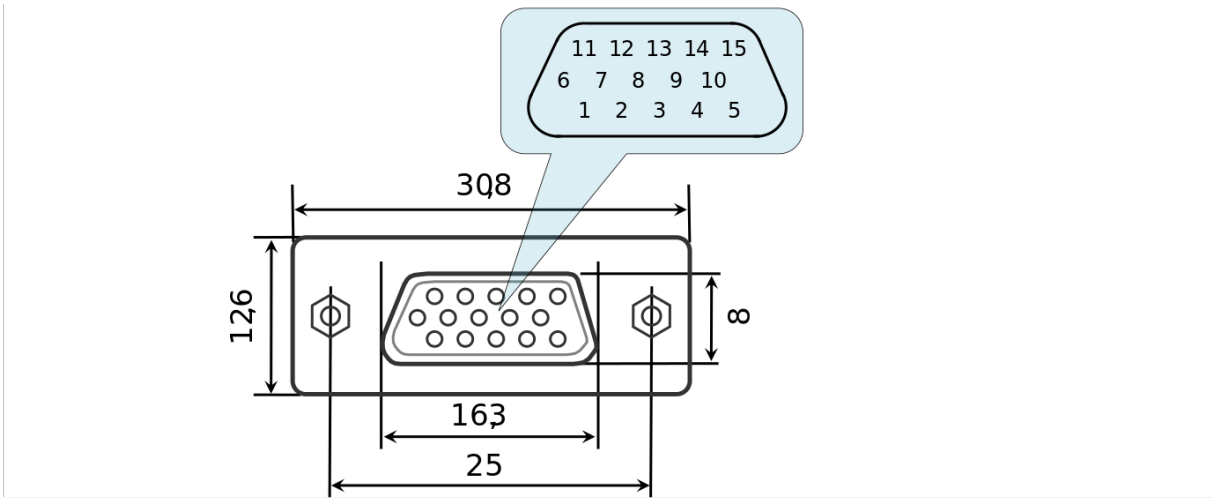


Fig. 4.9: Dimensions and numbers of the pins in DSub connector (front view)

Pins functionality:

1. Not phase B of SM or DC- of the motor
2. Phase B of SM or DC+ of the motor or phase B on BLDC motor
3. Not phase A of SM or DC- of the motor or phase C on BLDC motor
4. Phase A of SM or DC+ of the motor or phase A on BLDC motor
5. 5V output, 500mA - for 8SMC5, stabilized output for encoder power supply
6. ID, one-wire interface for stage identification (only for Standa stages)
7. Logic GND for limit switches, encoder, etc.
8. 2nd limit switch
9. 1st limit switch
10. Encoder channel A
11. Encoder channel B

12. Revolution sensor input
13. Inverted Encoder channel A
14. Inverted Encoder channel B
15. Inverted revolution sensor input

Note: Only firmwares 4.1.0 and older support BLDC.

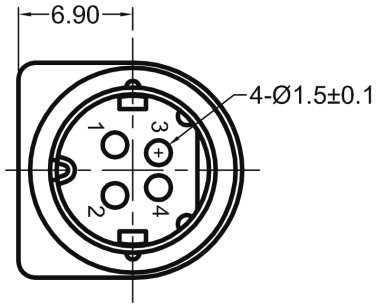
Note: Outputs 1 & 3 and 2 & 4 must be connected together for proper DC motor function if the nominal current of the motor is higher than 3A.

Warning: Plugging in/out the motor to the controller is not recommended while motor windings are under voltage.

4.1.3.2.2 Power supply connector

One and two-axis controller models use Kycon 4-pin DC power connector (part number KPPX-4P, www.kycon.com).





Pinout:

1. Power, "-".
2. Power, "+". 12-48V.
3. Power, "-".
4. Power, "+". 12-48V.

Important: Never supply the power to the controller and do not plug it to power connector if you are not confident that your power supply parameters conform to the requirements. Never attempt to plug the power supply to the controller if you are not sure power supply unit and controller connectors are compatible! The acceptable connection parameters are described in *Safety instructions*.

Important: Hot-swapping or unreliable connection of the power supply connector Kycon may damage the PC and/or the controller. For more details please refer to *Safety instructions*.

4.1.3.2.3 Data connector

Controllers connect via USB type-B connector.



Fig. 4.10: USB type-A - USB type-B cable

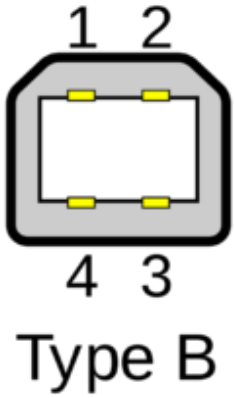


Fig. 4.11: USB type-B connector

Table 4.3: Output pin table

Pin #	Name	Wire colour	Description
1	VCC	Red	+5V DC
2	D-	White	Data -
3	D+	Green	Data +
4	GND	Black	Ground

Warning: Use verified USB cables only! Damaged or low-quality USB cable may cause improper controller operation, including motor rotation errors and errors of device recognition by PC operating system. Short cables with thick wires and screening are ideal for sustainable connection.

4.1.3.2.4 Joystick connector

A single-axis or two-axis controller model may contain a 9pin DSub male joystick connector.

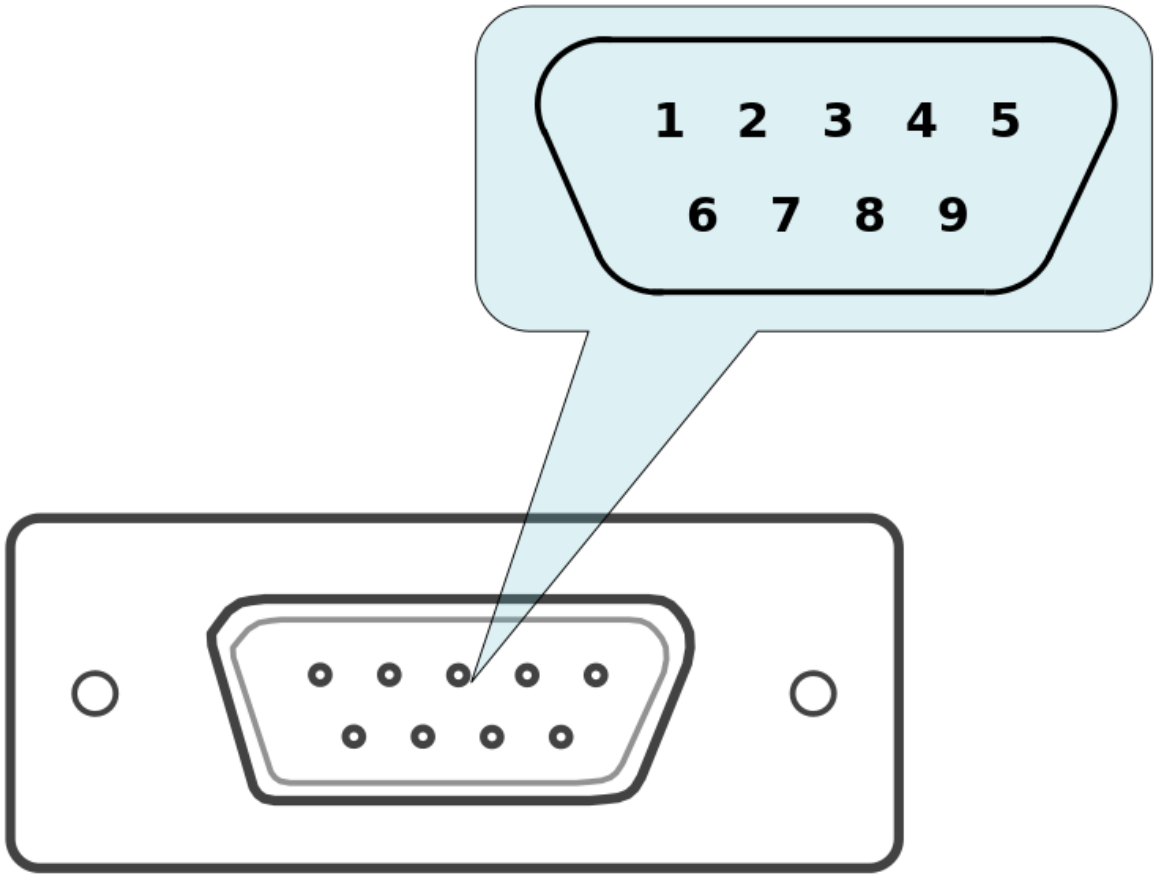


Fig. 4.12: Pinout of the joystick connector, front view

Pinout:

1. BUT_R_1, *right button* input, axis 1.

2. Joy_2, analog 0-3V *joystick* input, axis 2.
3. GND, common ground.
4. BUT_R_2, *right button* input, axis 2.
5. GND, common ground.
6. Joy_1, analog 0-3V *joystick input* input, axis 1.
7. 3.3V output.
8. BUT_L_1, *left button* input, axis 1.
9. BUT_L_2, *left button* input, axis 2.

Note: If the connector belongs to a single-axis model, then pins 2, 4, 9 are not used.

Note: Unused pins of the internal connector do not require any additional connection or pullup/pulldown. Simply do not use them.

Important: Analog Joy, Pot inputs are designed to work with LESS THAN 3V voltage. Do not apply higher voltages, including 3.3V, to these inputs, as it can break all analog controller inputs and lead to the controller or motor failure.

4.1.3.2.5 Supplementary two-axis system connector

Two-axis controller model contains a HDB-26 female DSub connector.

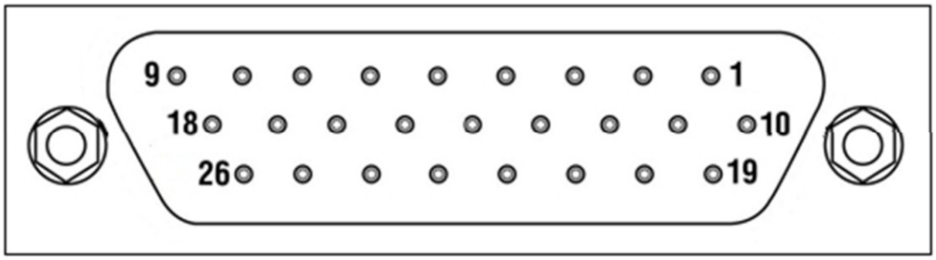


Fig. 4.13: Pinout of the supplementary HDB-26 connector, front view

Pinout:

1. AXIS_2_RX, serial port input for axis 2
2. AXIS_2_SYNC_IN, synchronization input for axis 2
3. AXIS_2_TX, serial port output for axis 2
4. DGND, digital ground
5. AXIS_2_SYNC_OUT, synchronization output for axis 2
6. AXIS_1_SYNC_IN, synchronization input for axis 1
7. AXIS_1_SYNC_OUT, synchronization output for axis 1
8. AXIS_1_RX, serial port input for axis 1

9. NC, not used
10. AXIS_2_CLOCK, clock signal for the external driver, axis 2
11. AXIS_2_POT, *analog input* for axis 2
12. AXIS_2_EXTGPIO_0, *input-output pin* for axis 2
13. AXIS_1_DIR_1, direction signal for the external driver for axis 1
14. DGND, digital ground
15. +5V
16. AXIS_1_EMBRAKE, brake control output for axis 1
17. AXIS_1_CLOCK, clock signal for the external driver, axis 1
18. AXIS_1_TX, serial port output for axis 1
19. AXIS_2_DIR_1, direction signal for the external driver, axis 2
20. AXIS_2_EMBRAKE, brake control output for axis 2
21. AXIS_2_PBRK, magnetic brake output for axis 2
22. PGND, power ground
23. AXIS_1_PBRK, magnetic brake output for axis 1
24. +5V
25. AXIS_1_EXTGPIO_0, *input-output pin* for axis 1
26. AXIS_1_POT, *analog input* for axis 1

4.2 Kinematics and rotation modes

4.2.1 Predefined speed rotation mode

Predefined speed rotation mode is the main operating mode of the controller for all the motor types. It allows to maintain the predefined rotation speed at a distance from the destination point and is usually used simultaneously with *Rotation for predefined point* or *Predefined displacement* modes. This mode may also get called by left and right motion commands.

Motor steps or *encoder* counts (if the encoder is available) per time unit are used as speed measurement units. Encoders work with all the types of motors. For DC motors the fixed speed is maintained even with the variable external load.

Turning on the *Acceleration mode* temporarily deactivates the *predefined speed rotation mode*.

After the controller receives the command to start the motion, it rotates the motor with user-defined speed. **The speed adjustment is available** at the *appropriate section of the "Settings..." menu of XILab software* or using the *set_move_settings()* function (refer to *Programming guide* section). For stepper motors the **speed value** may be defined in full steps and microsteps per second, for DC motors the speed is defined in revolutions per minute (RPM).

The speed for special motion modes, e.g., for *backlash compensation* or *automatic zero position calibration*, is different from the general rotation speed and is set separately.

The controller allows limitation of the maximum speed if appropriate parameter is defined by user. In that case any rotation that would have happened with the speed over the maximum is performed with the maximum speed. A separate adjustment is available, providing use of the maximum speed for all the ordinary motion modes, except for special ones, e.g., *backlash compensation* or *automatic zero position calibration*. The maximum speed adjustment as well as the adjustment for modes using this speed are available in the *appropriate section of the "Settings..." menu of XILab software*.

The **actual speed** is displayed in *XILab main window*, in the *Speed* field, or on *main operating parameters charts*.

Note: If the **stability of the speed maintenance** seems to be insufficient while the *encoder* is used, please refer to *recommendations for accurate rotation*.

Note: Maximum allowed speed is **100000 steps/s** or **100000 rpm** depending on engine type.

4.2.2 Rotation for predefined point

The *rotation to predefined point* mode is the main operating mode for all the types of motors and is usually used simultaneously with *predefined speed rotation mode*. It provides moving the stage to the defined position with **absolute value** for destination point coordinates which is different from *predefined displacement mode*. An additional reciprocal motion close to predefined point may be performed at *Backlash compensation* mode.

While using the *encoder*, few barely noticeable "vibrations" are possible before the motor stops in predefined point.

Besides, the movement to a given point, for stepper motors, can be carried out in the feedback mode *encoder mediated*. In this case, the movement is carried out in several iterations with position control at the end of each iteration of the encoder, until it hits a given coordinate with a certain accuracy.

After the starting command is received by the controller, it either switches on the *acceleration mode* (if the appropriate option is on) or immediately starts rotating the shaft of the motor with user-defined speed. After the predefined point is reached, the rotation stops; *deceleration* may be activated if the appropriate option is on. The **destination point** is set in *XILab main window*. The **destination point** may be defined either in full steps and microsteps for stepper motors or in *encoder* counts for all types of motors.

The **actual position** is displayed in *XILab main window*, in the *Control* section, or on *main operating parameters charts*.

Note: If the **positioning accuracy** seems to be insufficient while the *encoder* is used, please refer to *recommendations for accurate rotation*.

4.2.3 Predefined displacement mode

The *predefined displacement mode* provides displacement of the stage for predefined value relative to zero position, if this is a first command since the controller started or relative to position reached by the motor after the previous commands are completed, i.e., the destination point coordinate is a **relative value**.

This mode is useful when the absolute position is unknown or doesn't matter.

Note: The predefined displacement mode is activated either by corresponding command or by incoming synchronization pulse. For more information please refer to *TTL synchronization* chapter.

4.2.4 Acceleration mode

The *acceleration* function is active **by default**. The *acceleration* is used for smooth start or finish of the rotation without shocks that are inevitable when the predefined speed is reached instantly. Moreover, inertia of rotor and the other components of stage usually doesn't allow instant gathering of high speed which results in the loss of steps as well as failure in rotation during the operation without feedback. While operating the motor with feedback via encoder, the speed will increase as quickly as *motor limiters* allow. High acceleration makes the rotation unstable as well as

it makes more noise and vibration. That's why the acceleration mode is recommended. The *acceleration* function provides reaching of both maximum speed and sustainable motion even for motors with intermediate torque value.

The *acceleration/deceleration* mode works in the following way: during the speeding-up, while the required speed value is higher than the actual one, a gradual acceleration is performed at the *Acceleration* value which is measured in steps per squared second. After the required speed is reached, the controller switches to *predefined speed rotation* mode. Approaching the destination position, the controller begins to decrease the speed as the speeding-down would equal to *Deceleration* value and the motor would stop exactly at the destination point. Thus, this mode provides a trapezoidal speed profile. If the displacement distance is small, then the acceleration may change directly to deceleration; this will result to triangular speed profile. Turning the acceleration mode on and off, as well as setting of acceleration and deceleration value, is possible using XiLab software (see the *Settings of kinematics (stepper motor)* section) or by *set_move_settings()* command described in *Programming guide*.

The *Acceleration* value is adjusted independently from the *Deceleration* value – and there is a reason for it. Usually, due to friction effect resisting acceleration but contributing to deceleration, the maximum acceleration value is less than deceleration one. Therefore, for the fastest response of the stage either preset profiles should be used, or the acceleration/deceleration values should be established experimentally, according to what your stage may provide. For stepper motors working without feedback these are the values that do not lead to the steps loss. For motors with feedback the trapezoidal speed profile should be controlled using *XiLab charts*. The acceleration/deceleration values should be taken 1.5–2 times less than those resulting in speed profile distortion or step loss.

Note: Turning the acceleration/deceleration mode off is sometimes useful for multiaxis systems control where, during the motion along multi-dimensional paths, a continuous speed projection on each of the axes is required.

Note: The acceleration value is not displayed in *XILab main window*.

Note: Acceleration/deceleration values should be set as to allow the motor to reach target speed or decelerate from top speed to zero in less than **5 minutes.** If the acceleration/deceleration on *kinematic settings* page is set outside of this range, then the controller will return an "incorrect value" error and accelerarion/deceleration will be changed in controller to applicable value.

4.2.5 Backlash compensation

Backlash occurs in any mechanical device, e.g., in reduction gear or in worm-gear. Backlash results in differences in physical stage position when approaching the same point from different directions, whereas the motor shaft is exactly in the required position.

Backlash compensation mode is used in order to eliminate such ambiquity. Its activation allows user to determine the direction from where the stage would approach the destination point. Further on, the stage will approach the stop point from the defined direction only, eliminating the mechanical backlash. If the natural approaching direction doesn't match the selected one, then the controller drives the motor for some user-defined distance beyond the destination point and after that turns the motor around and completes the approach from the required direction.

While a loaded mechanical system is moving, its dynamic characteristics in the backlash zone do differ from the regular motion mode. Therefore, the rotation in the backlash zone should be performed with user-defined speed.

The following parameters of backlash-compensating system are available for adjustment by user:

- Backlash compensation on/off flag.
- Rotation speed while performing the compensating motion.

- Backlash compensation distance. The plus or minus sign for that parameter is used to determine the approach direction. The plus means the approach from the left side whereas the minus means the approach from the right side.

The controller indicates if the backlash compensation is active using MOVE_STATE_ANTIPLAY flag in the state structure which is also displayed in *XILab main window*.

A forced backlash compensation by using the *LOFT* command may be performed if there is no confidence that the actual position is backlash-free. While carrying out this command, a displacement for backlash compensation distance is performed with subsequent return. Calling this command while driving will lead to a smooth stop of the engine. This command makes sense only when the backlash system is active.

Note: The *backlash compensation mode* presumes no axis position correction, providing the user with just the choice of the direction from where the stage should approach the destination point, sticking to this selected direction.

The backlash compensation adjustment using XILab software is described in *Settings of kinematics (stepper motor)* section. Switching on and backlash compensation parameters detection commands are described in *Programming guide*.

The minimum backlash is reached if the approach to the setpoint is performed with the same movement parameters, so the optimal values of the backlash parameters are: the play speed must be equal to the nominal speed, the backlash compensation distance must be such that the device could reach the nominal speed.

Backlash compensation distance can be calculated from formula:

$$S = \frac{U^2}{2}\left[\frac{1}{Ac} + \frac{1}{Dc}\right] + 0.2U$$

S - backlash compensation, Ac, Dc - acceleration and deceleration, U - nominal speed, 0.2 - even motion time.

4.2.6 Rotation reversal

It is a common agreement that the coordinate increase corresponds to movement to the right, whereas its decrease corresponds to movement to the left. The rotation is to be reversed either if this rule is not satisfied due to physical stage location, or if the stage is supplied with an anchor which is pointed so that it doesn't match coordinate increase.

The rotation reversal may be switched on in the Motor parameters block of *XiLab menu*. Switching this feature on will change the current coordinate sign; thus, "left" and "right" terms will get interchanged. For example, the first movement during the *Home position calibration* will perform physically to the opposite direction, the *Left* and *Right* commands in *XILab main window* will interchange, etc.

Warning: Reverse is a setting that affects the whole controller operation if changed. The previously used *XiLab scripts* or *your own controlling programs* will work differently.

Particularly, the *limit switches* are adjusted independently from the reverse. Thus, after switching this mode on or off, one must re-adjust them.

4.2.7 Recommendations for accurate rotation

The controller can automatically adjust itself for the required mode, in order to maintain either the speed or the coordinate. However, both the speed and the adjustment property depend on the controller settings. The stepper motor working in steps and microsteps positioning mode can instantly reach the required operating conditions. If the stepper motor is physically unable to provide the required speed or acceleration, the rotation will most likely stop completely. The movement will not fail if a feedback sensor such as quadrature encoder is used as a reference; however, the controller probably won't be able to maintain the required rotation parameters.

The additional position sensor (encoder) is required for DC motors. The indirect connection of controlling scheme affection with DC motor stage displacement results to slowing down of reaching the required coordinate or speed. The following recommendations will help you to accelerate this process and to make it more stable:

- The profile corresponding to the stage being used is normally uploaded to controller and is used by it. Please upload the profile from the Standa website if you aren't confident that it is proper.
- The motor doesn't enter the limitation mode for one of the operating parameters (refer to *Motor limiters* and *Power control* chapters). Such limitations are displayed by the horizontal bar above the *Current* indicator in either *Power*, *Voltage* or *Speed* blocks in the *Motor* section of *XILab Main window in single-axis control mode*. For more information please refer to *Motor limiters* and *Power control* chapters.
- There are no mechanical impediments for rotation, the axis and stage are not jammed.
- The output power of power supply unit being used is sufficient (see the *Safety instructions*).

4.2.8 PID-algorithm for DC engine control

4.2.8.1 Algorithm description

DC engine is controlled by the PID regulator, with the coordinate as the controlled parameter. The controlled coordinate changes according to motion settings and incoming commands to provide motion capability. We will call controller coordinate the running position. DC engine winding PWM signal fill factor is the control signal of the regulator.

The control action is calculated according to the following formula:

$U(t) = I + P + D = K_P \cdot E(t) + K_I \int E(t)dt + K_D \frac{dE(t)}{dt}$, where:

$U(t)$ - is the control action

$E(t)$ - is difference between the running coordinate and the current motor coordinate

K_P, K_I, K_D - are proportional, integral and differential coefficients of the regulator. Regulator coefficients are set on *PID settings page* of the XILab program or programmatically by calling set_pid_settings() function of the libximc library (see *Programming guide*).

The resulting value is normalized according to the following formula to make PID regulator action independent of motor type, feedback sensor and working voltage:

$DC(t) = \frac{U(t) \cdot U_{nom}}{U_{supp}(t) \cdot IPS}$, where:

$DC(t)$ - is the PWM fill factor

U_{nom} - is the nominal (maximum) motor voltage, (see *Motor limiters*).

$U_{supp}(t)$ - is the current supply voltage

IPS - is the feedback encoder resolution in counts per revolution

This approach allows to change motor type, feedback sensor and voltage supply unit without reconfiguring PID regulator.

Warning: Do not forget to change *PID regulator coefficents* accoring to abovementioned formula if you are going to change maximum motor voltage.

4.2.8.2 Particular properties of the algorithm

4.2.8.2.1 PID regulator coefficients

User set values are normalized to keep optimal PID regulator coefficients in [0..65535] range.

Let's consider the effects different components have for better understanding.

We will assume the supply voltage $U_{supp}(t)$ is constant and equal to the motor nominal voltage U_{nom}. With this assumption PWM fill factor will be equal to 1 in the following cases:

1. $K_p = 1,\ K_I = 0,\ K_D = 0$ - if target position is ahead of real position by 256 motor shaft revolutions
2. $K_p = 0,\ K_I = 1,\ K_D = 0$ - if integral in the formula above is equal to 52.5 revolutions · second
3. $K_p = 0,\ K_I = 0,\ K_D = 1$ - if real motor speed is higher than the required speed by 96000 rpm.

4.2.8.2.2 Reaching target position

Target position is considered to be reached when motor shaft reaches the target position. Some oscillations around target position are possible. Motor will need some time to stop and return to correct position if smooth deceleration is not used and an immediate stop command is received or an emergency stop by limit switch has happened.

Warning: Long time oscillations around the target position while the motion is considered finished are possible if the PID regulator is set up incorrectly.

4.2.8.3 PID regulator tuning recommendations

There are three quality criterions used when tuning PID regulator:

- Speed maintenance accuracy, defined as a mean deviation of current speed from desired speed. Speed maintenance is considered to work optimally if the speed takes no more than three distinct values when moving. It is impossible to achieve greater accuracy because speed values are quantized.
- Position reach quality, defined by the following criterions:
  - Time until final stop in target position (lower is better).
  - No slip on approach to the target position.
  - No oscillations around target position before stopping.
  - No spontaneous shifts from the target position after stopping.
- Low noise when moving. Noise level is increased when only one of the PID coefficients is increased.

One can choose individual criterion priority depending on the task at hand when tuning PID regulator.

1. It is recommended to disable all DC motor limits including smooth acceleration/deceleration when you start tuning PID regulator.
2. It is recommended to tune PID regulator to maintain constant speed first.
3. It is recommended to start tuning PID regulator with integral and differential coefficients set to zero.
4. Set proportional coefficient $K_P \leq 10$
5. Set required movement speed and start the movement far away from the limit switches.
6. Gradually increase K_P and observe current speed graph. Find optimal K_P which makes the controller maintain current speed the best and makes time to reach target speed less dependent on K_P. One should be aware of possible noise level increase.
7. It is recommended to tune K_I next. Integral PID regulator coefficient has more influence on position reaching behavior.
8. Set integral coefficient $K_I \leq 10$ and start movement to position. It is convenient to use "shift on offset" command for this.

9. It is advisable to use sufficiently long shifts so that motor reaches at least 20% of required speed.
10. Increase of K_I makes the controller stop at the target position faster.
11. Achieve faster reaching of the target position by gradually increasing K_I. One should stop increasing this coefficient when either quality of position reaching degrades substantially or noise levels increase. Slip at the target position will remain regardless.
12. Adjust K_P up or down to decrease oscillations at the target position.
13. After one adjusts K_P one should check the speed maintenance quality and noise levels. If these parameters are unsatisfactory then one should adjust coefficients towards their previous values.
14. After setting K_I and K_P you can set up K_D.
15. It is recommended to start tuning K_D in speed maintenance mode and then check other quality parameters.
16. One should increase K_D until speed oscillations around target speed stop decreasing.
17. In case of target position slip one should increase K_D. However, further increases of this coefficent will lead to oscillations around target position. One should keep a balance between oscillations and target position reaching speed.

Turn on smooth acceleration/deceleration if you plan on using this mode. This can lead to the slip at the target position. To compensate one should increase K_I.

After these steps the initial setup of PID algorithm is complete. The obtained coefficients are in most cases suitable for motor operation. To further optimize these coefficients one can vary them while continuously monitoring quality according to the relevance of chosen quality criterions to the particular task.

Note: It is not recommended to adjust more than one coefficient at a time while tuning the PID regulator.

Important: It is extremely not recommended to start with large values of PID regulator coefficients or to rapidly adjust them. This can lead to self-excitation of speed oscillations and motor failure.

4.2.9 Feedback EMF

4.2.9.1 Advantages

- Always supports sinusoidal current form, which ensures silent operation;
- At high speeds, it can dynamically adapt to external loads, current and voltage restrictions (automatically reduces speed);
- At low speeds, it uses frequency control without rotor position control. When the position of the rotor begins to give correct indicators, it switches to the field control mode with feedback on the rotor position. The switching threshold is individual for each engine, it is determined by the quality of the rotor position assessment issued by the observer;
- Does not use the position sensor (encoder);
- It can operate in three modes:
  - **MTPA** - the most economical mode, characterized by a minimum current, but the voltage increases rapidly with the speed of $I_d = 0$;
  - **FW** - the flux linkage step-down mode is active when the set speed cannot be reached within the current voltage using MTPA $I_d < 0$;

– **Limit** - saturation mode, when movement at the specified speed is not possible. Occurs when the voltage and current are saturated. In this mode, the drive outputs the maximum torque determined by the current speed and current limits and sets the *PowerLimited flag*.

Important: The algorithm should not be used with the "Position Control" flag enabled. For smooth running in the EMF algorithm, a discrepancy between the actual position and the profile position is implemented. If "Position Control" flag is enabled, false Alarms may be triggered.

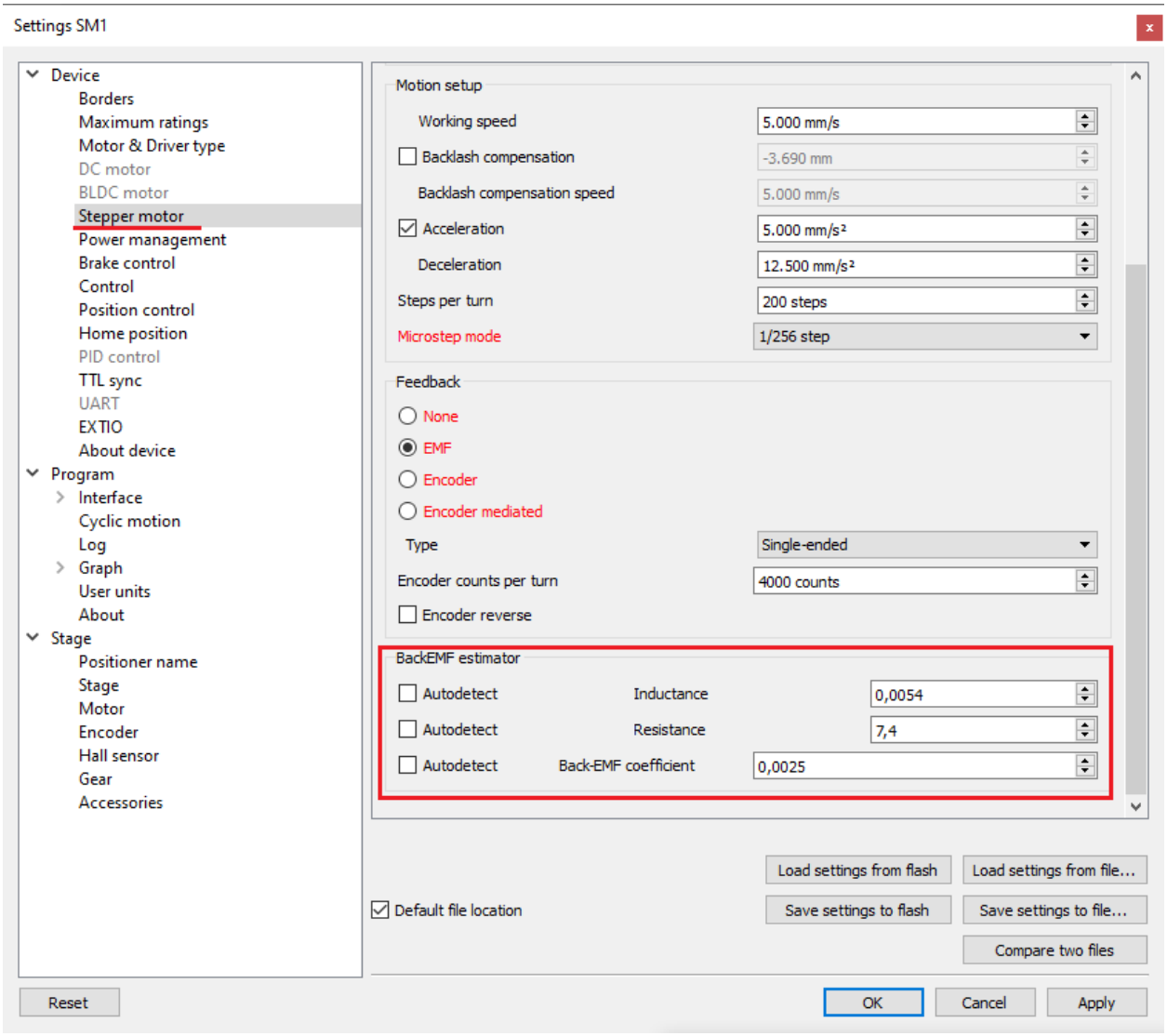
4.2.9.2 Behavior of the engine when exposed to an external force

In frequency control mode: - The rotor position is not controlled, but the current is equal to the nominal value. Only an external force greater than the holding moment can cause the steps to be lost

In field control mode: - If the force can be overcome, then the movement continues at the specified speed; - If the force cannot be overcome, the PowerLimited flag is set, and the set speed value begins to decrease in accordance with the Deceleration value, the position setpoint determined by the logic of the speed profile generator (integral of the speed) is changed accordingly; - If the force is stopped, the misalignment between positions will be compensated by the PID of the position controller; - If the force cannot be overcome by the drive (exceeds the holding force), then at the speed threshold there is a regular switching with subsequent rotor failure and loss of steps.

4.2.9.3 Selecting L, R, and backEMF parameters for EMF algorithm

After turning on the power (switching from the Power: Off state) and before starting to move, the parameters R and L are automatically detected (a short beep is heard). If these parameters are set via the xilab interface, the stage is skipped. To re-evaluate the parameters, the engine must be returned to the Power: off state.



On the *Stepper motor* tab, you can additionally assign the following parameters:

1. Resistance - winding resistance $R\,\Omega$
2. Inductance - winding inductance L H
3. Back EMF coefficient - the flux linkage of the rotor λ_m Hm/A
4. The value of all these parameters is saved in the engine profile.

Automatic detection of the *back-EMF parameter* works satisfactorily for most engines. However, assigning parameters via a profile provides greater stability of the algorithm.

The values $R\,\Omega$ and L H must be taken directly from the dataset.

The value of the rotor flux linkage λ_m Hm/a can be obtained as follows:

In the datasheet, the value of the motor's Electromechanical coefficient K_m (torque constant, Hm/A) or the counter-EMF coefficient K_{emf} (backEMF constant, Vs), then

$\lambda_m = \frac{4K}{n}$, where K is the value of K_{emf} or K_m, n - number of steps per revolution.

In the datasheet, the rated current i_n (nominal current, A) and the holding torque T_n (holding torque, Hm), then $\lambda_m = \frac{4T_n}{I_n n}$, where n - number of steps per revolution.

4.2.9.4 The choice of PID coefficients for EMF

In field control mode, when the rotor position estimation is available, the position is controlled using a standard PID controller. Its coefficients ensure the stability of the engine in the area of high speeds.

The accuracy of working out a position by profile is determined by many factors:

- In hold mode - the ratio of the hold force to the interference force is the same as for all open-loop algorithms;
- At low speeds - the discrepancy between the actual position and the profile position should not exceed one step;
- At high speeds - the discrepancy can be several steps, due to transients, feedback coefficients, and the presence of external forces.

We suggest using a standard set of coefficients: $K_p = 3.6, \quad K_d = 0.028, \quad K_i = 38$

- K_p increasing the coefficient increases the accuracy (reduces the θ_e error), reduces the adjustment time;
- K_d increasing the coefficient increases the damping of the system and reduces vibrations. A low coefficient can cause instability because the K_d value is too small;
- K_i has little effect on accuracy in transient modes, but it reduces the steady error when moving at a constant speed and acceleration, and also allows you to compensate for constant external forces. In many applications, it can be accepted as null. **If the value is too high, stability is lost**.

4.2.9.4.1 Operation algorithm

The controller **input** is:

- θ_r - desired rotor position (rad), the value of which is generated by the speed profile generator
- ω_r - the desired angular speed of rotation of the rotor (rad/s), the value of which is generated by the speed profile generator
- θ_m, ω_r - rotor position (rad) and speed (rad/c) calculated using the rotor position estimation

Output: I_{qr} - the current value (À) that determines the torque generated by the engine: $M = k_m I_{qr}$, where k_m is the Electromechanical coefficient (torque constant) of the engine

Parameters: K_p, K_i, K_d - feedback coefficients (values set in the *"PID control"* tab).

Control law: 1. Calculation of the control error: $\theta_e = \theta_r - \theta_m, \quad \omega_e = \omega_r - \omega_m$, 2. Calculation of current: $I_{qr} = K_p\theta_e + K_d\omega_e + K_i \int_0^t \theta_e d\tau$

The controller is equipped with an anti-accumulation loop based on the conditional integration algorithm. The growth of the integral part stops if the current saturation occurs ($|I_{qr}| > I_{max} and I_{qr} \cdot \theta_e < 0$)

4.2.10 Feedback encoder

This is the mode when all parameters of the engine including position and velocity are measured directly by the encoder and are denominated basing on counts of encoder. The position is displayed directly in the encoder counts, the speed is denominated in RPM (revolutions per minute). The speed is calculated by the controller basing on the speed alteration data as well as on the number of encoder pulses per one complete revolution of the motor shaft that are displayed in feedback configuration block at the Settings of kinematics (*Stepper motor*), (*DC motor*) folder. Note that in the case of DC motor the *speed maintaining mode*, the *mode of movement to the predefined point* as well as all their derivations use PID control algorithms and the *appropriate settings* are required. The driving encoder mode optimizes stepper motor control, this leads to noise reducing and facilitates a stable passage of the resonant speeds with no risk of steps loss when the coordinate flounders and the recurrent *calibration* is required.

The new algorithm is available with firmware 4.0.7+. Use XiLab 1.13.13+ and set Feedback to Encoder on the Device -> *Stepper motor page*. Note, that position is in encoder counts now.

4.2.11 Feedback encoder mediated

This mode of operation is optimally used in systems with large backlash to get to the desired coordinate in a few iterations, on average from 3 to 10. This mode is used for stepper motors.

For this mode, all motor parameters, including position and speed, are set and displayed directly by the encoder and have dimensions based on the encoder readings. However, in the controller the movement itself is carried out in steps.

Principle of operation: The coordinate values obtained from the external interface are converted into steps and the first iteration of the motion is performed,upon completion of which the position of the encoder is checked. Thereafter the deviation values for the new iteration are determined and a new entrance cycle is carried out. It occurs until the moment when it hits the specified coordinate with a given accuracy.

The new algorithm is available with firmware 4.3.+. Use XiLab 1.16.+ and set Feedback to Encoder mediated on the Device -> *Stepper motor page*.

4.2.12 Stop motion modes

There are two stop motion modes in the controller:

- immediate stop;
- stop with deceleration.

4.2.12.1 Immediate stop

Immediate stop is initiated by the command *STOP*. The controller tries to instantly stop the rotation of the motor shaft. This can lead to missed steps in a stepper motor, if no feedback is used. Abrupt cessation of movement can adversely affect the equipment, for example, may shift samples on the microscope stage or require additional adjustment of the optical line after a sudden stop.

> **Warning:** When the controller is configured to trigger a stop on the left/right *limit switch*, it always occurs immediately when the stage reaches the limit switch. This should be avoided.

4.2.12.2 Stop with deceleration

Stop with deceleration is initiated by the command *SSTP*. A smooth stop occurs with deceleration *Deceleration*, if it is not disabled in the *acceleration mode* settings.

> **Warning:** If the *acceleration mode* is disabled, then there is no difference between the immediate stop and the stop with deceleration. An *SSTP* command will then result in an immediate stop.

4.2.13 PID-algorithm for BLDC engine control

4.2.13.1 PID-algorithm description

BLDC engine is controlled by the PID regulator, with the coordinate as the controlled parameter. The controlled coordinate changes according to motion settings and incoming commands to provide motion capability. We will call controller coordinate the running position. Output current is the control signal of the regulator. The control action is calculated according to the following formula:

$$U(t) = I + P + D = K_p \cdot E(t) + K_i \int E(t)dt + K_d \frac{dE(t)}{dt}, where:$$

$U(t)$ - is the control action

$E(t)$ - is difference between the running coordinate and the current motor coordinate

K_p, K_i, K_d - are proportional, integral and differential coefficients of the regulator. Regulator coefficients are set on *PID settings page* of the XiLab program or programmatically by calling set_pid_settings() function of the libximc library (see *Programming guide*).

The effects different PID components (Kp, Ki, Kd) have are same for BLDC and DC motors. See *PID-algorithm for dc engine control*.

4.2.13.2 PID regulator manual tuning

We provide a special XiLab extension for the manual adjustment of the PID regulator coefficients. The time dependence of the speed of the BLDC engine and the speed retention error is shown in a special window, see the screenshot below.

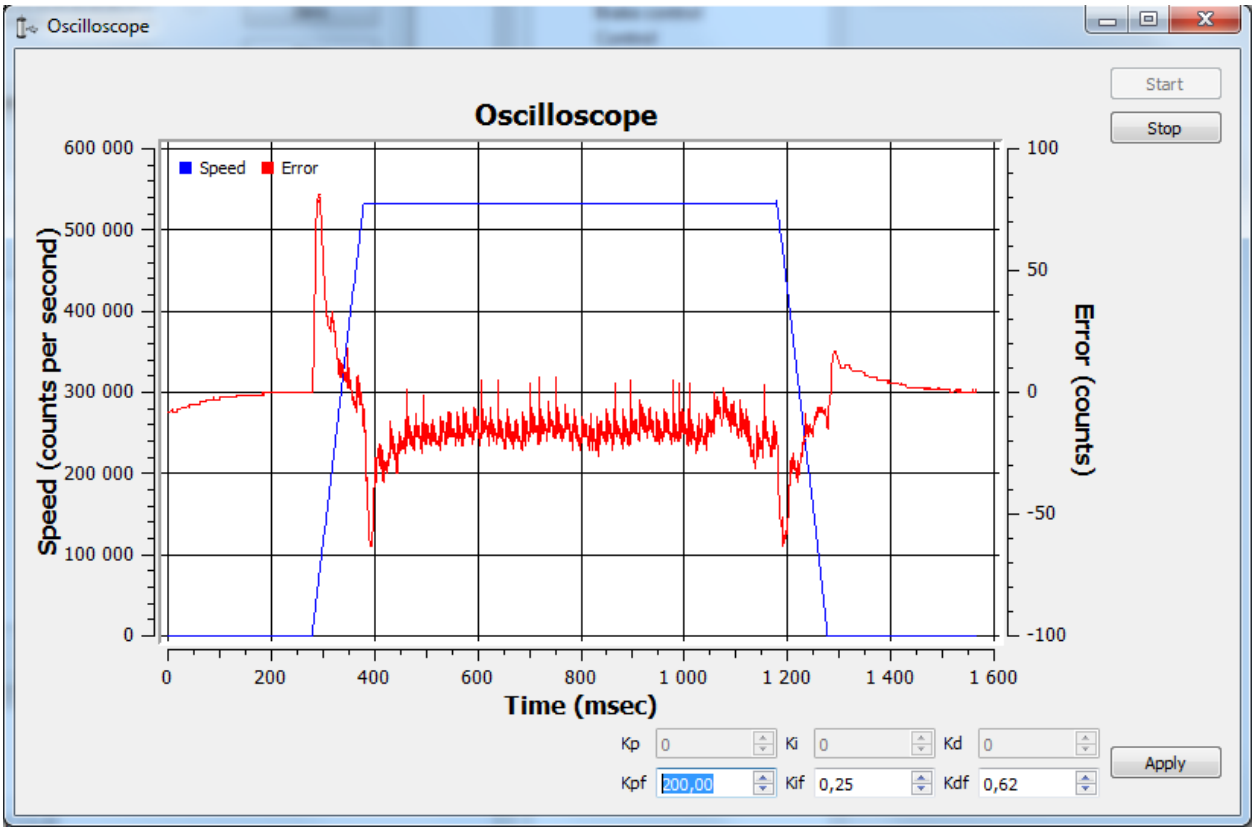


Fig. 4.14: The PID regulator tuning window.

The stable position retention is necessary for the correct engine operation.

4.2.13.2.1 Steps to adjust the coefficients:

1. First, you need to evaluate the PID coefficients. Given the structure of the managed system, they can be calculated from simplified formulas. For this, the parameters from the documentation for the appropriate motor and stage are used.

    - K_m - electromechanical motor coefficient [H / A] (the torque generated by the current strength is 1 A). Can be calculated as the ratio $K_m = \frac{F_n}{I_n}$, where F_n is the nominal (maximum) force generated by the

motor, I_n is the rated (maximum) current strength.

- M - weight of load (kg).
- $\sigma = \frac{M}{K_m}$.
- $K_p = 11500\sigma \cdot 1000,\ K_d = 186\sigma \cdot 1000,\ K_d = 12.2\sigma \cdot 1000.$

2. Set the coefficients calculated by formulas, click Apply. Click the Zero button on the *main XiLab window*. Set 0 to the Move to field, send the command. The engine should stop. Try to move the position manually, make sure that the response is correct - the engine tries to return to zero position (the encoder reverse is set correctly).
3. Set a small speed in the *motion settings*, click Apply. Start moving in the *main window*. The differential coefficient (Kdf) should be increased if there are vibrations and disruptions.
4. If the vibrations have audio frequencies (the stage emits a loud sound when driving), it may be necessary to reduce the Kd coefficient or all the coefficients proportionally.
5. The integral coefficient (Kif) is responsible for getting into the target position, it is convenient to use the command Shift on for testing.
6. To fine-tune the coefficients use the Oscilloscope window where the speed retention error is displayed for used motion parameters.

Note: Only special XiLab version support oscilloscope window. Contact us to get it.

7. After the coefficients are adjusted, they need to be proportionally increased/decreased, this corresponds to an increase/decrease in mass, response to the impact becomes more/less powerful, sudden stops will not lead to disruption of movement.

4.3 Main features

4.3.1 Supported motor types

Currently the controller supports stepper and DC motor types. The parameters of supported motors are described in *Specifications* chapter.

4.3.1.1 Stepper motors

Rated current is the main parameter of the stepper motor. The rated current is adjustable at the *Settings of kinematics (stepper motor)* section.

Important: The motor will gradually overheat and get physically damaged if rated current is exceeded. Make sure that the rated current value is set according to the used stage. All the settings are proper in default stage profiles.

Step division mode is another important parameter. In full step operation, the motor moves through its basic step angle (for example, a 1.8° step motor takes 200 steps per motor revolution). Microstepping can divide a motor's basic step up to 256 times. Microstepping improves low speed smoothness and minimizes low speed resonance effects.

The following step division options are available:

- 1 (full) step
- 1/2 of the step
- 1/4 of the step
- 1/8 of the step

- 1/16 of the step
- 1/32 of the step
- 1/64 of the step
- 1/128 of the step
- 1/256 of the step

The microstep mode is set either on *Settings of kinematics (stepper motor)* page or by motor adjustment commands. See the *Communication protocol specification* and the description of the related functions in the *Programming guide* chapter.

Note: The controller always uses the internal step division value equal to 1/256. If the user selects a coarser step division, the software will display only the multiple of the coarser step division positions and both adjustment and transmission are possible only in that coarser step division mode. This is done for compatibility with both obsolete and actual software operating with small multiples of the step division values. On the other hand, operations with the largest step division value provides the most smooth and silent rotation at the smaller speed values.

The number of steps per revolution is the another direct parameter of the stepper motor. This setting does not affect the rotation but is used in *slipping control* block or with motors with the *encoder* feedback.

Note: The controller supports stepper motors with feedback sensor - encoder. The *encoder* can be used as the main position sensor (*more info*) and as the slippage, backlash or steps loss detector (*more info*). Using the encoder facilitates a stable passage of the resonant speeds without movement disruption.

4.3.1.2 DC motors

Unlike stepper motors, controlling DC motors requires feedback. Currently only *encoder* is supported as a feedback sensor.

Main DC motor parameters are maximum current and voltage, which can be set on *Settings of kinematics (DC motor)*. Main *encoder* parameter is counts per revolution.

Important: The motor will gradually overheat and get physically damaged if rated current is exceeded. Make sure that the rated current value is set according to the used stage. All the settings are proper in default stage profiles.

Important: Setting wrong value of encoder counts per revolution will lead to the controller being unable to maintain speed correctly. In some cases it may lead to stage or reducing gear failure.

DC engine is controlled by the *PID regulator*. Please carefully read *PID-algorithm for DC engine control* before you start working with it.

Important: Wrong PID regulator settings might lead to stage failure. All supplied profiles are preset with correct PID settings. It is not recommended to alter these settings unless absolutely necessary.

4.3.1.3 BLDC motors

Firmwares 4.1.x and older support BLDC motors controlling. Like DC motors, controlling BLDC motors reqiures feedback.

Main BLDC motor parameters are maximum current and number of poles, which can be set on *Settings of kinematics (BLDC motor)*. Main *encoder* parameter is counts per revolution.

Like DC, BLDC engine is controlled by the *PID regulator*. Please carefully read *PID-algorithm for BLDC engine control* before you start working with it.

Important: Like DC motors, wrong PID regulator settings, current and encoder settings might lead to stage failure

4.3.1.4 Engine selection criteria

Pulse-width modulation (PWM) is a widely used way to control winding's current in different motor types. It leads to current oscillations at PWM switching frequency (so-called "current ripple"). Current ripple's amplitude depends on motor characteristics like its winding inductance and resistance. Motor can heat up more than it is expected with nominal current due to current ripple, i.e. $\frac{P_{real}}{RI_s^2} > 1$. There is RI_s^2 - power dissipated by I_s (stabilization current), P_{real} - real power, dissipated in motor. For overheating estimation we recommend to use this graph:

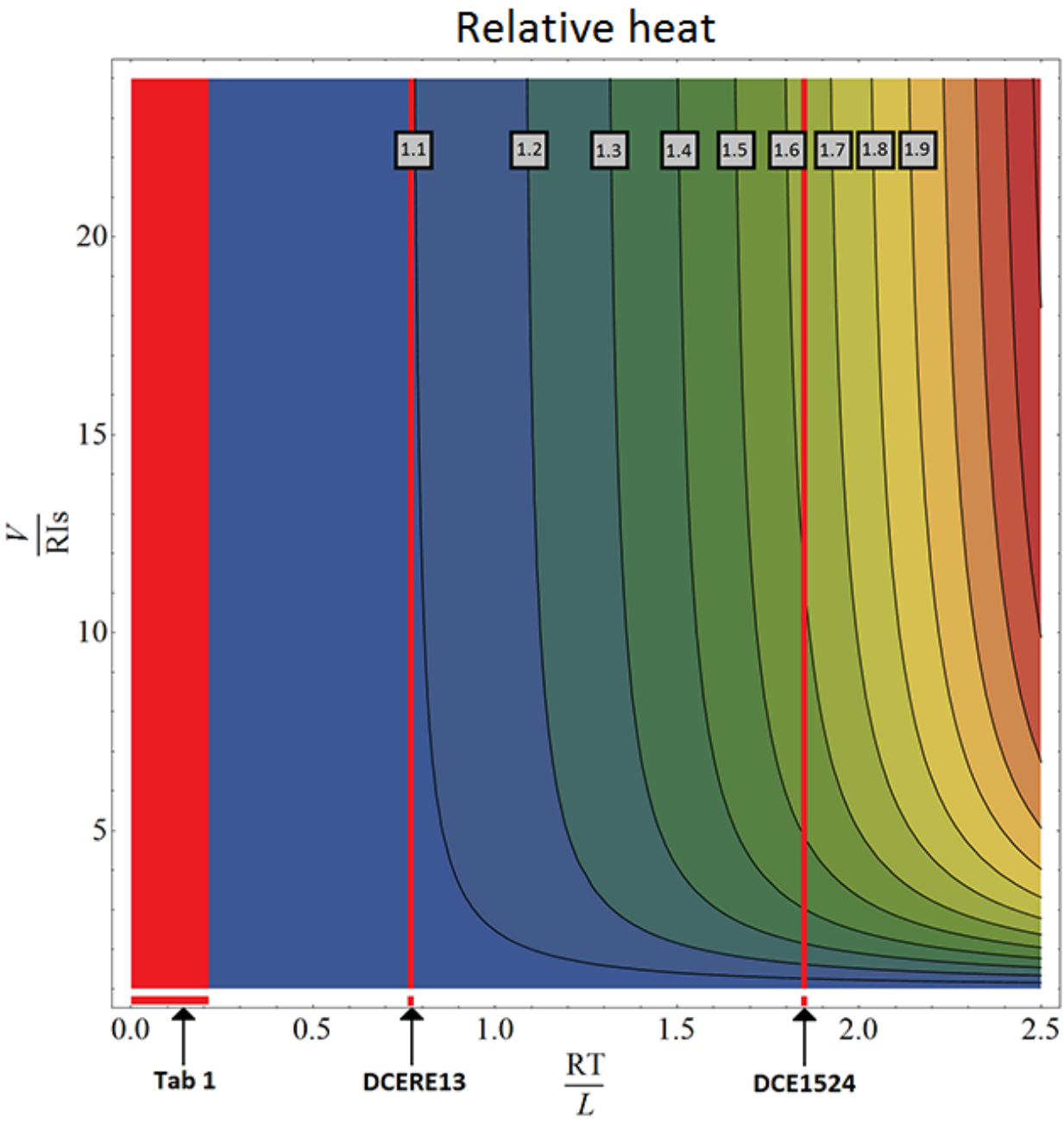

Relative heat
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
20
15
10
5
V/RIs
0.0
0.5
1.0
1.5
2.0
2.5
RT/L
Tab 1
DCERE13
DCE1524

Table 4.4: RT/L values for some motors

Motor	RT/L
20	0.19576
28	0.07253
28s	0.07168
4118L1804R	0.02715
4118S1404R	0.02844
4247	0.0273
D42.3	0.0223
5618	0.0146
5618R	0.0146
5918	0.0116
5918B	0.012
VSS42	0.029
VSS43	0.0256
ZSS	0.04248
DCERE25	0.2106

The motor's overheat is determined by this sequence:

- $\frac{RT}{L}$ calculation. There is R, L - resistance and inductance of motor winding (refer to the motor documentation). T - PWM period time. It's value should be 51.2 us for stepper motors and 25.6 us for DC motors.
- $\frac{V}{RI_s}$ calculation. This parameter shows power voltage excess under nominal voltage. There is V - power voltage,:math:*R* - winding resistance, I_s - stabilization current.
- Overheat definition. After first two steps point can be plotted at the graph. Now we should define the region, which corresponds to overheat degree. For example, the region between 1.1 and 1.2 corresponds overheat value between $1.1 * RI_s^2$ and $1.2 * RI_s^2$.

There is DCE1524 overheat calculation example:

- T = 25.6 us
- R = 5.1 Ohm
- L = 70 uH
- $\frac{RT}{L}$ = 1.86

Now we can draw vertical line corresponds this value (look at graph) and find out overheat with different power voltages. Let's assume I_s = 500 mA. Nominal voltage in this case is $R * I_s$ = 2.55 V. If power voltage exceeds nominal more than 5 times but less than 10 times DCE1524's overheat will be between 1.5 and 1.6. Motor overheats about 1.65 times with 30 V of power voltage.

All the major engines and their parameters have been marked in *RT/L values for some motors*.

4.3.2 Motor limiters

Motor winding current and voltage limiters and motor shaft revolution speed limiters are provided to ensure safe operation. These limiters, if activated, will lead to gradual power and rotation speed decrease until the parameters being limited are within acceptable range. Motor limiters work with voltage and current values directly on motor windings, unlike *critical parameters*, which work with current and voltage at the controller input. Another distinctive feature of limiters is that they do not stop the motor and let the controller enter **Alarm** state; they merely limit voltage, current and motor revolution speed.

For DC motors:

- *Max voltage* is the nominal motor voltage. It defines maximum motor winding woltage. It is usually used to limit voltage growth when the motor or stage is jammed. *Should only be used if maximum motor winding current is unknown.* This parameter is used in *PID regulator*.
- *Max current* defines maximum engine winding current. It is usually used to limit current growth when the stage is jammed. One should set this limit based on maximum current which can be sustained by the motor without damage (primarily by heating).
- *Max RPM* is the maximum motor shaft revolution speed. Is usually used to limit revolution speed when working with reducing gears and other devices which have strict maximum speed limits.

Note: One should be aware of the distinction between maximum motor current and nominal current. In general they may be different because of motor cooling features and operating conditions. Also, one should not mix up maximum current and starting current with stationary motor shaft.

Important: Changing maximum motor voltage might disrupt PID regulator. For more information see *PID-algorithm for DC engine control*.

Important: Maximum motor voltage may exceed nominal voltage, usually by 10-15%. If you are using a motor with low load and you need high motor speed, then you can increase maximum motor voltage.

Current limiter operation.

It is important to note that maximum current limiter does not react immediately when working with DC motors. When a higher than maximum current is detected in the motor winding, voltage supplied to the motor is gradually decreased until winding current is less than *Max current*. In the worst case of a rapid jam during high-speed movement motor voltage may decrease for at most 370ms. If the current limit is chosen right, motor should not overheat during this time.

Note: If the *Max current* value is set too low, it is possible that DC motor will not be able to move under high load or high friction.

For stepper motors:

- *Max(nominal) Speed* is a maximum motor shaft rotation speed in steps per second. Current stepper motor speed is defined by *Speed* parameter (see *Predefined speed rotation mode*).
- *Nominal current* defines maximum motor winding current. This value cannot be exceeded due to characteristic stepper motor control.

In the XILab software limiter settings are described in sections *Settings of kinematics (DC motor)* and *Settings of kinematics (stepper motor)*.

4.3.3 Limit switches

4.3.3.1 Limit switches designation

Limit switches are designed in order either to prevent the stage movement out of permissible physical movement range or to limit its movement range according to user-defined requirements. Incorrect setting of the limit switches may result to stage jam if the controller goes beyond the permissible range.

https://youtu.be/OICjZNFeBV8

Differences between mechanical and optical limit switches. This video clearly shows how mechanical and optical switches affect the accuracy of homing and how the position "floats away" as the homing repeats in the xilab software.

4.3.3.2 General settings

If the limit switch is active, a corresponding flag is placed in the state structure and the appropriate icon (left or right) is displayed in *XILab Main window*. The controller can either stop any movement in the direction of any active limit switch (left or right) or stop the movement to the single limit witch (left or right) or not to limit the movement. Limit switches settings are performed in XILab software (see the *Motion range and limit switches* section).

4.3.3.3 Programmable motion range limitation

If there are no hardware limit switches for the motion range but the stage requires such limitation, the programmable limiters can be used. For doing that, the limiters should be switched to limitation mode according to position reading (see the *Motion range and limit switches* section). The left and right margin fields are used (the right margin value should be higher than the left one). In this mode, the left limit switch is active if the actual position is less than the left margin value and the right one is active if the actual position is greater than the right margin value. The operation time is about one millisecond.

Warning: The programmable motion range limitation is reliable only if there is no direct setting of the new position by ZERO or SPOS commands, or if there is no steps loss or encoder malfunction if it is used for positioning, or if there is no frequent power-cut during the rotation. If any of these problems appears, the programmable range should be re-adjusted. The appropriate reference sensor allows the automatic re-adjustment using the *automatic Home position calibration* feature.

4.3.3.4 Hardware limit switches

The controller may operate with limit switches based either on dry contacts, or on optocouplers, or on reed switches, or on any other sensor types generating a 5V TTL-standard "logic one" electric signal in one state and a "logic zero" in the other. Each limit witch may be configured independently. There is also possible to change the position of limit switches or their polarity in software.

Note: Limit switches are also useful for *automatic Zero position calibration*.

4.3.3.5 Limit switches connecting instructions

Limit switches should be connected to *D-SUB connector* pins as it is shown at the diagrams:

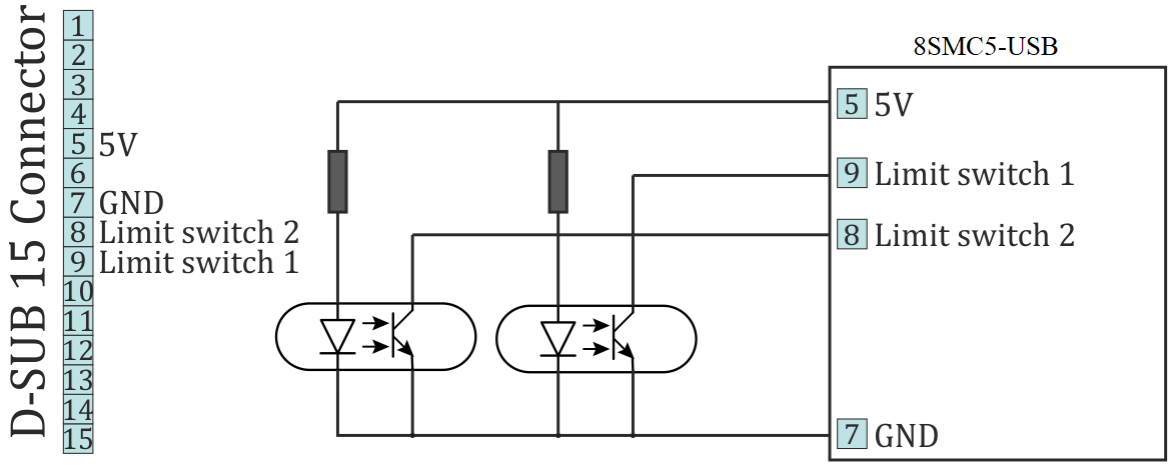


Fig. 4.15: The “optocoupler” limit switches connection diagram

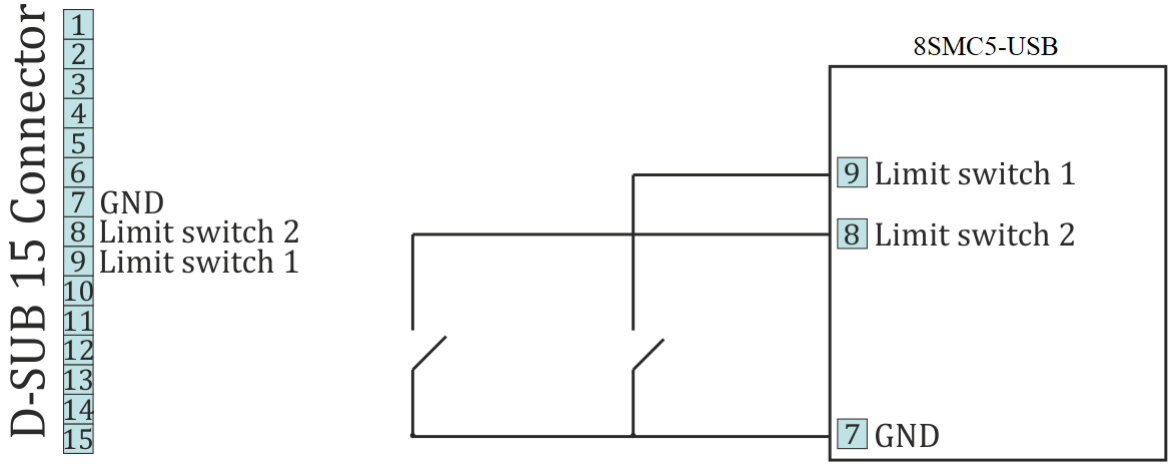


Fig. 4.16: The “dry contact” limit switches connection diagram

4.3.3.6 Limit switches location on translators

The settings of which limit is left or right is required by the controller. Sometimes it is unknown a priori, just it is clear that both limit switches are connected and fire if the corresponding limit of the motion range is reached. The stage jam is possible if the limit switches are configured improperly. Therefore, the controller supports just a simple detection of incorrectly configured limit switches, shutting down the movement on both of them. Please make sure that their polarity is configured correctly and the shutdown mode is activated on both of limit switches. The flag of improper limit switch connection detection should be turned on in corresponding *XiLab software menu*. Start the movement to any direction until the limit switch shuts the movement down. If there was right-side movement but the left limit switch became active, or vice versa, the limit switches should be interchanged (see the *Motion range and limit switches* chapter). If the improper actuation of the limit switch is detected and if the corresponding feature is set in the *Critical parameters* menu, the controller can turn the Alarm mode on.

Warning: The protection against mistaken limit switches connection doesn’t guarantee the complete solution of the problem, it only makes the initial configuration procedure easier. Particularly, don’t start the movement if any of the limit switches is active, even if the protection is on.

4.3.4 Automatic Home position calibration

Autocalibration (homing) is used for detection and placing the movement to the starting position (it can also be called "home" or "zero" position). Calibration comes to automatic accurate detection of the *limit switch*, the *revolution sensor* signal or moment of getting the *external synchronization pulse* which determines the zero position, and grading from it by a specified offset. This allows one to start working in a situation when the current position of the stage is unknown, but the location of one reference point (initial position) relative to the limit switch or some other signal is known. The homing process does not require programming skills from the user.

The reference point (stop signal) is determined in one of three ways, depending on the settings selected by the user:

- Movement until the *limit switch* is reached - in this case, the current settings of the limit switches (location, polarity) are used. For more details please refer to *Motion range and limit switches* chapter.
- Movement until a signal from the *revolution sensor* is received, in this case the current settings of the revolution sensor are used. For more details please refer to *Position control* chapter.
- Movement until the signal from the *synchronization input* is received, in this case the actual configuration of synchronization input is used. For more details please refer to *Synchronization settings* chapter.

Warning: If the synchronization input is software-disabled in the appropriate settings menu, then the signal from it will never be processed.

4.3.4.1 Standard homing algorithm

Depending on the settings the homing can be processed using three different algorithms. The standard search for a home position is the following: the controller starts moving with the preset parameters of speed and movement direction until a stop signal is received. The speed of homing is usually set lower than the working speed in order to "not miss" the arrival of the stop signal and improve the calibration accuracy. Then an absolute offset is made at a working speed to the preset standoff distance.

The resulting point is called the starting position or "home position". It is important to note that its location on the stage does not depend on the initial position from which calibration started.

4.3.4.2 Accurate additional calibration

After the stop signal is received the position of the controller is already determined. But before making a shift to the home position one can perform the additional movement towards the next stop signal (the second phase of homing). This allows to reach an accuracy in setting a home position of 1/256 steps for stepper motors or 1 encoder count for DC and BLDC motors for some stages. If the corresponding flag is set, the controller rotates the motor in a user-defined direction with the preset speed until a stop signal is received from the source selected by the user. Then, as for the standard algorithm, the offset is made at a working speed to the preset standoff distance.

The parameters of the second phase of homing (speed, movement direction and the source of the stop signal) are set independently of the first phase parameters settings. At the same time it is reasonable to use the signal from the revolution sensor placed on the motor shaft previous to the gear and perform movement at a low speed - this will provide the maximum accuracy. Since the stop signals for the first movement and the second movement can coincide, a special flag is provided in the software to start tracking the stop signal for the second movement only after making a half turn of the motor shaft. This avoids the ambiguous sequence of receiving stop signals for the first and second movements. As a result of the optional second movement the calibrated position is refined.

Note: In case the second phase of homing is used the first movement can be performed at high speed since it only roughly calibrates the position and accuracy is not required there. The accuracy will not be increased in case the second limit switch is used for the second phase of motion since its physical parameters do not differ from those of the first limit switch.

4.3.4.3 Fast homing algorithm

When the fast homing is enabled the controller starts to rotate the motor towards the preset direction with a working speed in order to quickly find the position of the reference point. After the stop signal is received the controller pulls the motor back for half a turn and starts moving again in the preset direction but takes the speed value specified in the settings of the first phase of homing. After the repeated stop signal is received the offset is made at a working speed to the preset standoff distance. The stop signal source is also set by the standard homing algorithm settings.

The fast homing algorithm is optimal for most stages and stages and as a rule is set by default in the profile settings for most of the Standa stages.

4.3.4.4 Autocalibration features

Successful completion of home position calibration results in assertion of the STATE_IS_HOMED flag in the *controller state structure*. In case the home position is somehow lost after the calibration (stop on *limit switch*, *immediate stop* while moving, *the detection of loss of steps*, switching to the *alarm mode*), the corresponding flag is dropped and it is necessary to re-calibrate the home position.

Note: The position reached as a result of calibration will slightly depend on the speed of the last motion until the selected sensor responsed. Therefore, don't change the speed parameters for further successful reaching the same position.

Note: If command *immediate stop* or command *power shutdown* are executed while the engine is stopped then it isn't necessary to re-calibrate the home position and the STATE_IS_HOMED flag is not dropped.

The description of the **functions** for auto-calibrating a home position is given in the *Programming guide* chapter.

Autocalibration commands are described in the *Communication protocol specification* chapter.

Autocalibration can be configured by the user in the XILab program on the Device tab -> Home position (see the section *Home position settings*), and started with the **Go home** button in the *XILab main window*.

A **set_zero** *script* is supplied with XiLab software pack, providing the automatic home position configuration. This script changes the Standoff setting at *Home position settings* tab, making the actual position as the Home one.

How to use the script:

- place the movement to the desired position,
- launch the script and wait until it's finished.

As a result, the stage will be moved in the same position where the set_zero script was called and all the following calls of homing function will move it there. Make sure to *save the settings* to controller's nonvolatile memory.

4.3.5 Operation with encoders

4.3.5.1 Application of encoders

Encoders are designed for creation of accurate and fast feedback according to the coordinate for all the electric motor types.The feedback is performed by the motor shaft position, by stage's linear position, by the motorized table rotation angle or by any other parameter related to the shaft position and measured by using the two-channel quadrature encoder complying the requirements described in *Specifications* chapter for the appropriate controller type. Controller **8SMC5** supports differential encoders and simple (single-ended) encoders..

Warning: Auto-detect works only with 3.3V and 5V (with error 0.2V) encoders.

4.3.5.2 What is quadrature encoder?

Encoder is a mechanical motion sensor. The quadrature encoder is designed for direct detection of the shaft position. The sensor transmits the relative shaft position by using two electric signals at CH A and CH B channels shifted relative to each other at 1/4 of period.



Fig. 4.17: The signals at CH A and CH B outputs of quadrature encoder

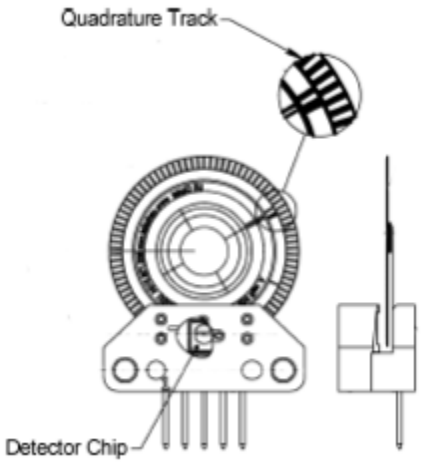


Fig. 4.18: An optical quadrature encoder mechanics

An optical quadrature encoder mechanics is shown at the figure above. There are two optocouplers used. The operational principle of an optocoupler is as following: a LED and a detector are arranged opposite to each other from different sides of a disc. The optocoupler opens when disc's "window" coincides with the detector (the outgoing signal is logic zero). The outgoing signal is logic one if the detector is closed by opaque part of the disc.

Number of steps per revolution (CPR) is the main parameter of the quadrature encoder. The standard resolution values for encoder are from 24 to 1024 CPR. Each period of signal alteration is interpreted by 1, 2 or 4 codes which is corresponding to X1, X2 and X4 operating modes. This controller uses the most accurate X4 mode. The maximum frequency of each encoder's signal depends on the applied encoder itself, since for 200 kHz in X4 mode the controller can read up to 800,000 encoder counts per second.

4.3.5.3 Controller's features

There are two operating modes with encoder available for the controller:

- the encoder is used as the main position sensor (this is the main operating mode for DC motors and the optional one for stepper motors).

- slippage, backlash or steps loss detection (the recommended mode for joint operations with stepper motors, in case the encoder is not used as a primary position sensor, see *more*).

4.3.5.4 Encoder connection

The encoder is connected to the controller via *D-SUB 15 pin* connector, which is in all systems (*controller board*, *one-axis* and *two-axis* in box

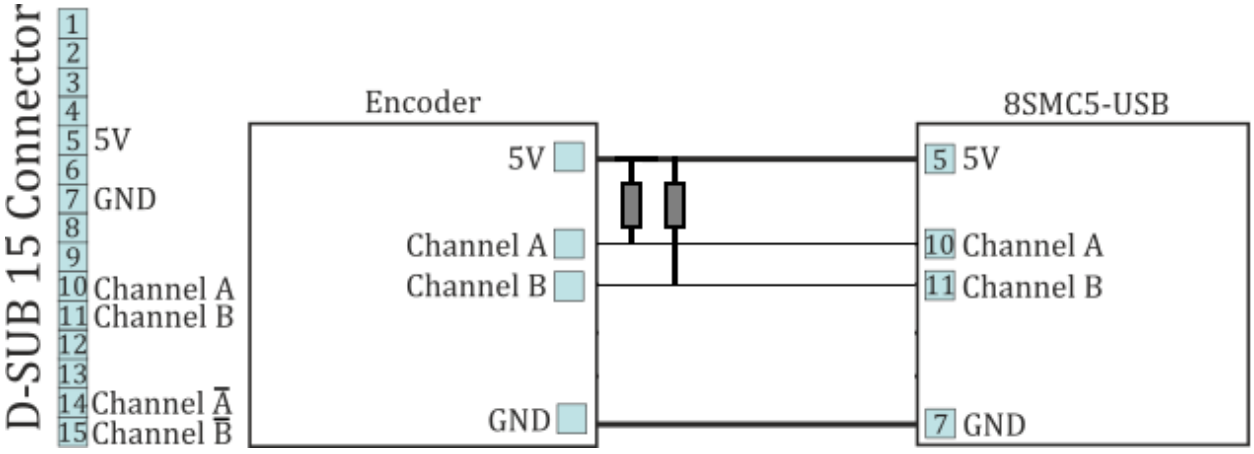


Fig. 4.19: The diagram of single-ended encoder connection using D-SUB 15 pin connector.

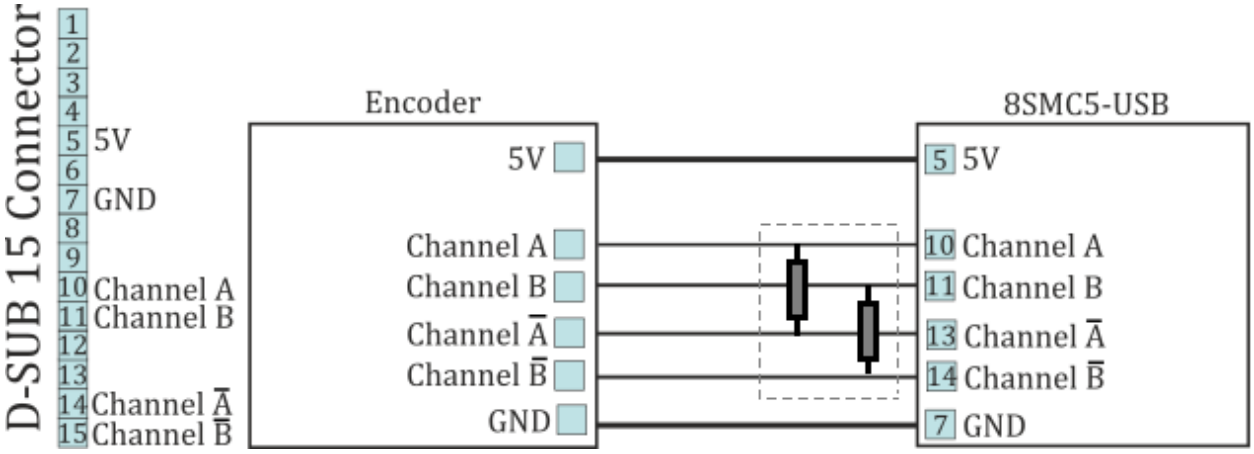


Fig. 4.20: The diagram of differential encoder connection using D-SUB 15 pin connector.

See also the *Example of a motor connection* chapter.

Warning: Encoder inputs of the controller internally pulled up to logic one by using the 5.1kΩ resistors. Frequently encoder outputs are of "open collector" type equipped with internal pull-up resistor. During the data transmission they provide good characteristics while passing from higher logic level to lower. However, the pass from logic 0 to logic 1 is more graduate. It passes through the RC circuit formed by pull-up resistor and cable capacitance. This is the most important thing if the cable is long *(up to 5 meters)*. If the internal pull-up is not sufficient, the pull-up resistor with r=1.5kΩ may be added for every +5V to each output in order to improve the transmission speed parameters; before doing that please check if the open collector of the encoder can transmit

5mA current. The resistors insertion diagram is shown above. The maximum operating speed for quadrature encoder may be increased by adding a push-pull driver with the outgoing current over 10mA to its output, providing quick 0 - 1 and 1 - 0 transmission edges.

4.3.5.4.1 Operation with long cables

For correct encoder operations on cables longer than 5 meters, it is recommended to use an encoder with a differential output (RS485) to reduce the effect of electromagnetic interference. When using the RS485, all differential pairs must be terminated with a 120 Ohms resistor in the connector to the controller.

The cable must have an additional internal shield for digital signals (pins 5-15) connected to the DGND (pin 7) on the controller side and on the stage side. The external shield must be connected to the metal case of the connector directly on the side of the stage and to the metal case of the connector through a 47 nF capacitor on the side of the stage.

4.3.5.4.2 Automatic encoder type detection

The 8SMC5 controller can automatically detect the type of encoder *if the corresponding option is enabled*. This system is designed to work with standard CAT-5E cables up to 50 meters long and with a conductor resistance about 80 mOhm per meter. Automatic detection may not work well with cables longer than 50 meters or with non-standard cables with high resistance wires. In case of problems with automatic encoder type selection, encoder type can be selected manually in *the feedback settings*.

4.3.6 Revolution sensor

Revolution sensor is designed for *stepper motor shutdown (failure) detection* and for better accuracy of Home position calibration procedure (see *Automatic Home position calibration*).

The controller may receive the actual position data from the external revolution sensor mounted on the stepper motor shaft. The sensor transmits its signals to the controller once or many times per one revolution of the motor.

Usually the revolution sensor is a small disc with precise graduation scale mounted on the motor shaft. A light source (LED) and a sensor of the optocoupler are placed at the opposite sides of the disc. The sensor is open if there is no interrupter between the LED and the sensor (the logic zero is transmitted to optocoupler's output), whereas the logic one is transmitted if the light source is closed by the interrupter.

By default, the lower logic level is interpreted by the controller as the active mode of the revolution sensor. The controller's input is pulled to logic one level, thus, the disconnected revolution sensor means its inactive mode. The controller's input can be inverted if necessary, in that case the logic one level will mean the active mode.

4.3.6.1 Connection diagram

The revolution sensor should be connected to the controller via *15pin D-SUB* connector, which is in all systems (*controller board*, *one-axis* and *two-axis* in box).

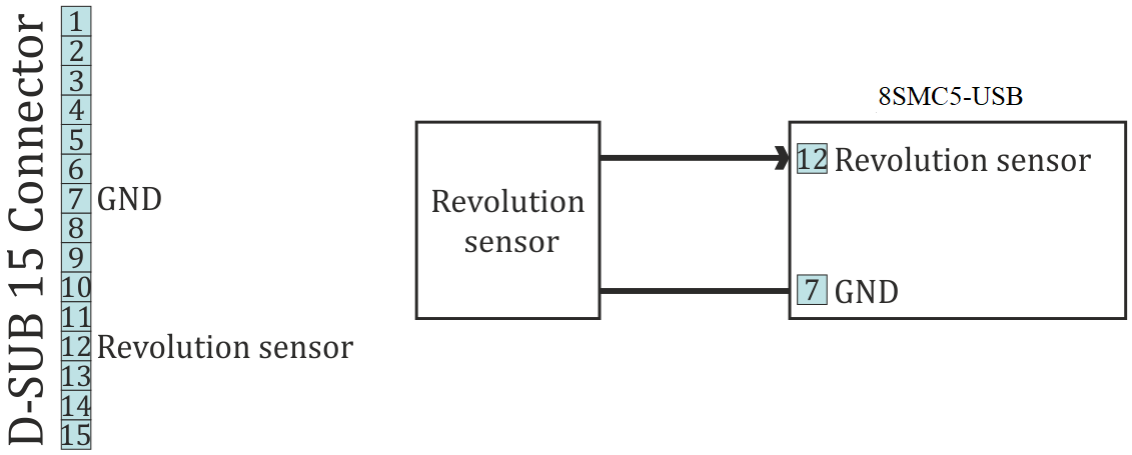


Fig. 4.21: Scheme of revolution sensor connection to the 8SMC5-USB based system

4.3.7 Steps loss detection

This mode is generally used while operating the stepper motor at full speed or limit loads when the shaft jam resulting to the steps loss is possible. In this case an additional position sensor (*revolution sensor*) or *encoder* allows tracking this moment, informing the user about it. This feature should be applied **with stepper motors only** and it allows detection of the steps loss. Steps and microsteps are the measurement units for all coordinates and shaft positions.

When the encoder is used, the controller stores both number of steps and number of encoder's counts per revolution (see the *Settings of kinematics* folder of XiLab program). When the feature is activated, the controller saves the current position in steps of the motor and the current position according to the encoder data. Then, during the motion, the position data according to the encoder converts to steps and if the difference exceeds the predefined value then the slippage is indicated and the *Alarm mode* turns on (if the related option is active). For more information regarding use of encoder as the steps loss detector please refer to *Operation with encoders* chapter.

If the revolution sensor is used, the position is controlled according to it. The controller stores the current position in steps according to active and inactive edges at the sensor's input. Then, at every revolution (number of steps per one full revolution is set by *Steps per turn* parameter, see the *Settings of kinematics (stepper motor)* chapter) the controller checks if the shaft has been displaced and how many steps for. If the mismatch exceeds the predefined *Threshold* value (which is defined in position control settings, see the *Position control* chapter), the slippage is indicated by the state structure flag. If the appropriate flag is set and if the error is detected, the controller turns the *Alarm mode* on and the motor shuts down, otherwise the motion is continued. If the slippage indication flag is active, the controller turns the *Alarm mode* on when the appropriate parameter in the settings is active.

Also you can enable the position correction option in the *position control settings*. If this option is enabled and the steps loss is detected the controller stops the movement, adjust the step position on the basis of the encoder data and try to start the movement again. The flag of the control position error *The position control error flag* is set when the desynchronization of the steps and the encoder position is detected and it will be unset automatically when the position becomes corrected. If the controller is not able to eliminate the desynchronization the controller is set *the position control error flag* and goes to *the Alarm mode*. If the steps loss happens during the movement *the movement command status* will not be changed while the position is correcting. If the steps loss happens during holding a position *the move to position* command will be executed for return the motor axis to the holding position.

Note: For using the position correction function you should have the encoder with the resolution at least two counts per the motor step.

Note: For correct operation of the position correction option you should let the controller to hold the position during

1 second for calibration before moving. It necessary to repeat the calibration after the transition to *the Alarm mode* or after changing the settings.

Note: If the automatically position correction is used it is not recommend to set the *Threshold* value above than 3 steps because in this case not any slippage will be corrected.

Note: *The soft stop* and *the hard stop* commands could be ignored by the controller if it was sent during position correction process. In this case you can send *the soft stop* command twice for power off the motor windings.

Note: If you use *the software limit switches* it is not recommend to use the automatic position correction because the limit switches positions will be changed during position correction process.

Note: A hard STOP launches the the re-calibration process of the revolution sensor position, and the calibration starts after the revolution sensor activates during the motion controlled by the motor. It means that the slippage won't be detected if the shaft has been rotated manually right after the hard stop since the calibration hasn't been performed yet.

Note: If the motor revolution sensor is bouncing (mechanically), the misoperations of the revolution sensor are possible at the very low speeds.

Note: The position control of the revolution sensor can't detect the shaft rotation at the zero speed, i.e., if the motor is shut down and the shaft is rotated manually, it won't be detected.

4.3.8 Power control

4.3.8.1 Current consumption reduction

Controller has an option to set current when idle to reduce power consumption. This mode is active by default. It is widely used to lower stepper motor heating in hold mode while keeping position maintenance accuracy. Hold current is set as a percentage of nominal winding current. A time delay after which current will be reduced is also defined. Current reduction mode can be disabled. To set current reduction see *set_power_settings* function in *Programming guide* or XILab *Power consumption settings* page. Nominal engine current is set by *set_engine_settings* function (see *Programming guide*) or on *Settings of kinematics (stepper motor)* page in XILab.

A reasonable hold current level is 40-70%. This will lower power consumption 2-4 times, while keeping holding force sufficient. A reasonable time to reduce power lies in 50-500 ms range. This is a sufficent time for mechanical oscillations, which might knock the system out of the hold position, to subside.

4.3.8.2 The motor power shutdown

There is also a power shutdown mode to reduce power consumption of a stepper motor. It is mostly used to stop wasting power on position hold, when no movements are performed for a long time. This mode is on by default, but can be disabled by the user. Time from motor stop to power off is set in seconds. A reasonable time is 3600 seconds (one hour). To set power off options see *set_power_settings* function in *Programming guide* or XILab *Power consumption settings* settings page.

4.3.8.3 Time delay calculation specifics

All timeouts work in the follwing way: on each transition to stop state time is saved with millisecond accuracy. After certain set time is elapsed depending on PowerOff/CurrentReduce enabled state a motor will reduce winding current or turn its power off. All settings can be changed online. For example, if you increase PowerOff timeout value after the poweroff has already happened then windings will get powered on and a PowerOff function will activate after the new delay. Timeout countdowns cancel after each movement start.

4.3.8.4 Jerk free function

Sometimes smooth motor winding current changes are required to reduce vibrations of a mechanical system. That's why a *Jerk free* option is provided, which allows one to set current ramp-up time from zero to nominal value with millisecond precision. When this option is turned on all changes to stabilization current or winding powerdown will happen with smooth current increase or decrease. For example, if jerk free time is set to 100ms and the controller needs to reduce current to 50% it will be reduced over the time of 50ms (because 100ms are required to reduce current from full to zero). To setup Jerk free see *set_power_settings* function in *Programming guide* or XILab *Settings of kinematics (stepper motor)* page.

Smooth current change function activates on any change in the amplitude of the winding current, for example on nominal hold current change. In this case current change speed is calculated based both on older and newer hold currents, whichever is higher. If controller needs to turn off the motor windings then current is gradually ramped down, then power output circuits are disconnected. If controller needs to power up the windings, then they are powered with zero current which increases up to nominal current.

There are exceptions to the rule, when the current is immediately reduced to zero even if Jerk free option is active. These are the critical errors/Alarm state (see *Critical parameters*) and controller reset events on firmware update. These events are rare and should not happen during normal stage operation.

A reasonable Jerk free time is 50-200ms, which merely leads to low-energy mechanical oscillations on 3-10 Hz frequencies which are significantly lower than noise from other common sources. Higher Jerk free times will lead to constant delays when current is switched on or off.

4.3.9 Critical parameters

Minimum and maximum values of currents, voltages and temperatures are used for safe controller operation. Any value out of acceptable range leads to the motion stop, windings power-down and Alarm state for the controller. Exiting the Alarm state is possble only after the critical parameter returns to normal and the STOP command is sent to the controller. Critical settings are used for all motor types.

The following parameters are available:

- *Low voltage off* defines the minimum voltage value of the controller power supply (measured in tens of mVs). The *Low voltage protection* flag is used to turn this option on, otherwise the minimum unpowering threshold doesn't work. The 6000mV to 8000mV range is sensible for operating power range of 12V to 48V. This type of protection helps to determine the power-cut moment due to activation of any sort of power supply unit protection. This may occur if the operating power consumption of the stabilized power supply unit is exceeded.
- *Max current (power)* defines the maximum current of the controller power supply (measured in mAs). The sensible value is twice the maximum operating consumed current registered during the tests. Use the *XiLab charts* for registration of the consumed current.
- *Max voltage (power)* defines the maximum voltage value of the controller power supply (measured in tens of mVs). The sensible value is 20% higher than power supply unit voltage.
- *Temperature* defines the maximum temperature of the microprocessor (measured in tenths of degrees Celsius). The microprocessor can operate at the working temperature of up to 75ºC and doesn't overheat by itself. Rise of its temperature indirectly indicates the overheating of the power part of the board. The overheating threshold range from 40ºC to 75ºC is sensible.

Flags:

- *ALARM_ON_DRIVER_OVERHEATING* means entering the Alarm mode if the driver's critical temperature (over 125ºC) is exceeded. The power driver indicates if its temperature is approaching the critical value. If the driver is still working then the further heating will automatically shut it down. It is recommended to set this flag and not to rely on automatic forced shutdown.
- *H_BRIDGE_ALERT* means turning the Alarm mode on if any fault of the power driver due to board overheating or damage is detected. This flag should be set on.
- *ALARM_ON_BORDERS_SWAP_MISSET* means turning the Alarm mode on if the triggering of the wrong limit switch, not corresponding to direction, is detected (see the *Limit switches* chapter). This flag is intended for clear indication of the response of the limit switch swap detection subsystem. The flag is recommended to be set on.
- *ALARM_FLAGS_STICKING* flag activates the sticking of the error indicators in the status structure of the controller, otherwise indicators are active only during the accident that caused the error. If there was a short-time error and its cause was independently removed, then sometimes the reason of Alarm remains uncertain. In that case the sticking is useful and the accident cause can get diagnosed in XiLab main window.
- *USB_BREAK_RECONNECT* - This flag configures the operation of an USB break reconnect block. When set, this unit starts to operate and monitor the loss of communication over the USB bus (for example, in case of a static discharge).

Configuration of parameters is described in *Critical board ratings* menu of XiLab software. The maximum available value configuration commands are described in *Programming guide*.

4.3.10 Saving the parameters in the controller flash memory

The controller provides an option to save all its parameters into the non-volatile memory. The configuration is restored when the controller is powered on, after that the controller itself is instantly ready for operation. The stage requires no new adjustment every time the power is on. The controller stores its user-defined name which is useful for its further identification.

The non-volatile memory stores all the actual operating parameters of the controller related to *Device* section of XiLab settings menu. Either **Save settings to flash** button of XiLab program or *command_save_settings* function are used for it (see the *Programming guide* chapter).

All the configuration parameters can get restored to controller's RAM from the non-volatile memory, not only when the power is turned on but also by clicking the **Load setting from flash** button of XiLab program which provides the access to the data saved in the flash memory. Internally it uses *command_read_settings* function (see the *Programming guide* chapter). The restored settings become active instantly and all the modules of the controller get re-initialized.

4.3.11 User defined position units

Controller position is set and read in stepper motor steps or encoder counts, if encoder is available and enabled. Is it convenient to set position in mm (in case of translation stages), in degrees (in case of rotator stages) or in any other natural units. Controller software can translate coordinates to user-defined units: user can set a ratio, where a certain amount of controller steps is equal to the certain amount of user-defined units. This enables one to issue movement commands and read controller position in these user units. It applies both to XiLab interface and to usage in custom programs or scripts. Speed and acceleration are also set in units derived from user-defined ones (for example mm/s). *Zero position adjustment* can be done the same way in user-defined units as in encoder counts or step motor steps.

You can enable user-defined units in *XiLab* on page *User units settings*. You can define the name of the natural units in XiLab.

Libximc library functions operating in user defined units have a _calb suffix. They take calibration structure calibration_t as an additional input. For more information see *Programming guide*.

4.3.12 Usage of a coordinate correction table for more accurate positioning

If a shift without a linear encoder is used, the exact position will not always be in correspondence with the indications of the axis coordinates. This is due to the accuracy of the manufacture of mechanical parts, backlashes, temperature expansion. In this case, you can use the correction table for more accurate positioning.

Important: The table is individual for each motion. The table is formed by the manufacturer on a high-precision stand.

Principle of operation:

After certain distances, not necessarily equal, starting with 0, the real position of the motion is measured. The difference between the specified and the actual position is recorded into the table. Based on the obtained values, with the usage of linear interpolation, the coordinates are recalculated with the help of certain _calb functions. As a result when moving, manufacturing inaccuracies and other possible position deviations are compensated.

Example: Suppose the following correction table is set for the stage.

X	0	5	10	15	20	25
dX	0	0.05	0.02	-0.003	0.01	-0.04

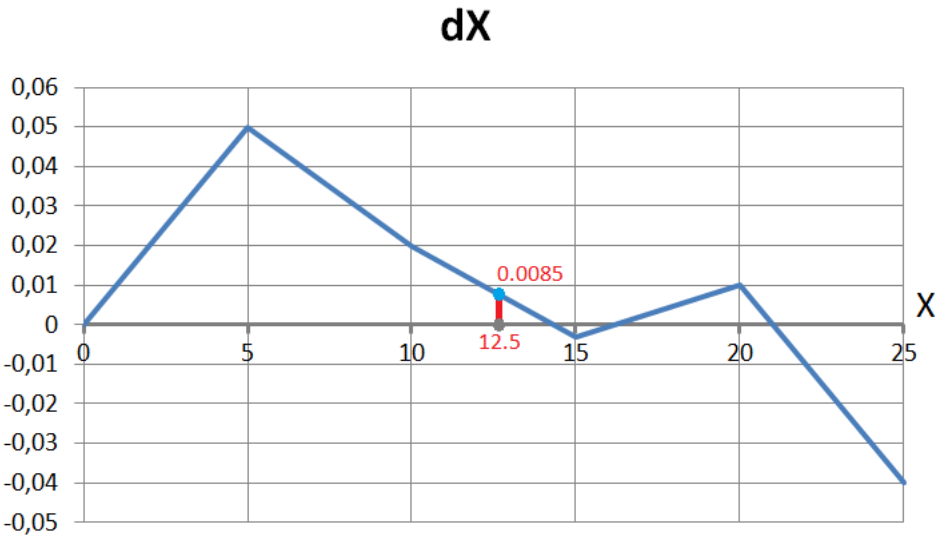


Fig. 4.22: The graph shows the coordinate deviations corresponding to the table

In order to move to position 12.5, the coordinate must be set to 0.0085 greater and that is 12.5085. This is exactly what the algorithms of some _calb commands that use the correction table do.

You can load and clear the coordinate correction table table in *XiLab* on page *User units settings*.

To see a list of functions, structures, and parameters that are corrected with the correction table, follow library guide libximc.

4.4 Safe operation

Several controller settings are directly connected with safe operation. If these settings are set wrong it may lead to controller or stage damage. Positioning element can be damaged by exceeded power, rotation speed, or by moving

outside the allowed movement range. Usually it is enough to load a preset profile for your stage for safe operation, where all necessary settings are already made.

4.4.1 Movement range bounds and limit switches

Linear stages have limited movement range, unlike circular rotators. Moving outside of the allowed physical movement range is the main reason for stage jamming and damage. To prevent such kinds of breakages the movement range of stages is limited according to user requirements. For this reason *Limit switches* are used, but in some cases when, for example, when stage is not equipped with limit switches or has only one limit switch, movement range can be defined in software (see *Limit switches*). Frequently *limit switches* are reversed. In this case use the mechanism of reversed limit switches detection which is described in *Limit switches* section because otherwise the first motion to the border will lead to stage jamming. *Motion range and limit switches* is described in corresponding section. Settings commands are described in *Programming guide*.

4.4.2 Movement range limiters

Nominal winding current is the main safety setting in stepper motors. This is the main parameter which defines power delivered to the motor. The nominal current should not exceed maximum allowable current for given motor. For more detailed information see *Motor limiters*. For DC engines nominal current is a limiting parameter and should be set according to the maximum permissible current through DC engine. If maximum current is not known, then maximal voltage delivered to the engine may be limited. This also will prevent engine overheat although voltage limiting is a more coarse mode than current limiting. For more detailed information see chapter *Motor limiters*.

Exceeding the speed limit might damage the stage or lead to faster wearout. It is necessary to set speed limit flag and to set correct maximum speed for the given stage. For more detailed information see chapter *Motor limiters*.

4.4.3 Critical Parameters

Controller tracks voltages and currents which appear in its circuits and can react on their suspicious values. This reaction blocks the engine and prevents any further movement until the source of the problem is eliminated. Due to this it is possible to track winding-winding or winding-ground shortcuts which may happen because of stage cable damage or damage of the stage itself. This reaction also has informational character because it allows to track incorrect values of source voltage or oncoming overheating. That's why you should read *Critical parameters* chapter and set necessary protection. In case of dangerous situation controller will enter Alarm state and the *main window of Xilab program* will be colored in red. If this happens, track and eliminate the source of danger before you turn off the Alarm. If you are using your own application for engine control you should pay close attention to Alarm status flag (see *Controller status*).

4.4.4 Operation with Encoder

If during encoder connection sensor channels are swapped, then during engine motion encoder will show direction in reverse. During operation with DC engine this situation will appear if engine control channels (DC+ and DC-) are mixed up. To fix these errors just set *Encoder Reverse* flag in Feedback section on *Settings of kinematics (stepper motor)* page for stepper motors and on *Settings of kinematics (DC motor)* page for DC motors.

It is also possible that there is no contact with one of encoder channels. In this case during motor motion values of sensor will be oscillating in [-1..1] range around the starting position.

During DC engine operation both of these errors will lead to malfunction in control algorithm, which is described in *PID-algorithm for DC engine control*. If you have connected new DC engine for the first time it is strongly recommended to check encoder connection before starting the operation. To do this you should set corresponding regulation factor values $K_P = 1, K_I = 0, K_D = 0$ and try to make motion to the right or to the left at a low speed. After the motion please check if encoder values are changing in correspondence with chosen directions. Set *Encoder Reverse* flag if it is needed.

4.5 Additional features

4.5.1 Operating modes indication

4.5.1.1 Controller status

Mode indication is provided in controller. For this purpose one dual-color LED is located on the board.



8SMC5-USB *8SMC5-USB-B8-1*

Green Power indicator shows presence of 3.3 V power supply of controller.

Red Status indicator represents controller operating mode. Simultaneous glowing of both lights looks like **yellow glow**.

Table 4.5: Power/Status indicator operation modes

Flicker frequency Hz	Description
LEDs don't glow	the controller is shut down, there is no power supply
Green Power LED is glow	power is supplied to the controller
Green Status LED is glow	the firmware is not loaded
Yellow Status LED is glow	the controller is in an *Alarm state*
Yellow Status LED is flashes, 0,25 Hz	the controller is operating, but there is no connection to the PC via USB
Yellow Status LED is flashes, 1 Hz	the controller is operating, waiting for the movement command
Yellow Status LED is flashes, 4 Hz	the controller is operating, the movement command is executed
Yellow Status LED is flashes, 8 Hz	the controller is in re-flashing mode
Yellow Status LED is flashes, 10 Hz	the controller is in USB bus reconnecting mode

4.5.1.2 Indication of limit switches

Multifunctional 18 pin BPC connector is provided with limit switches activity indication. High logic level appears on the corresponding output in the moment of limit switch activity. Active state is determined from limits switches settings (see *Motion range and limit switches*)

4.5.1.3 Connection diagram

Note: In case of additional LEDs they should be designed for 4 mA operational current. There is no need in additional current limiting resistors. Operational current for typical LEDs is about 2 mA. It is not recommended to use blue and violet LEDs because of their high cut off voltage and as a consequence low brightness.

4.5.1.3.1 Controller board

Status indicator is duplicated by output on the PCB connector CN1 *BPC connector* with the next scheme of LED connection:

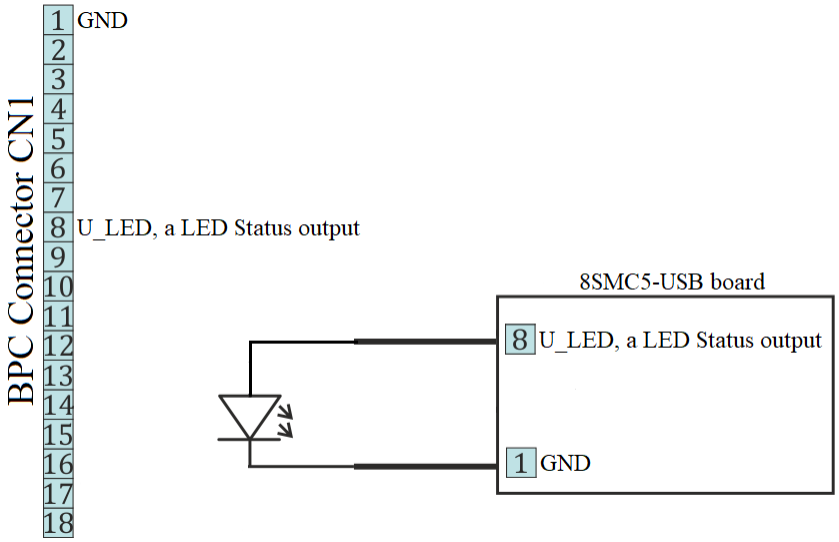


Fig. 4.23: Connection of Power and Status indicators to the controller board

Limit switches are located on the same connector. The connection diagram is shown below. To indicate the limit switches it is convenient to use the LEDs designed for the necessary current.

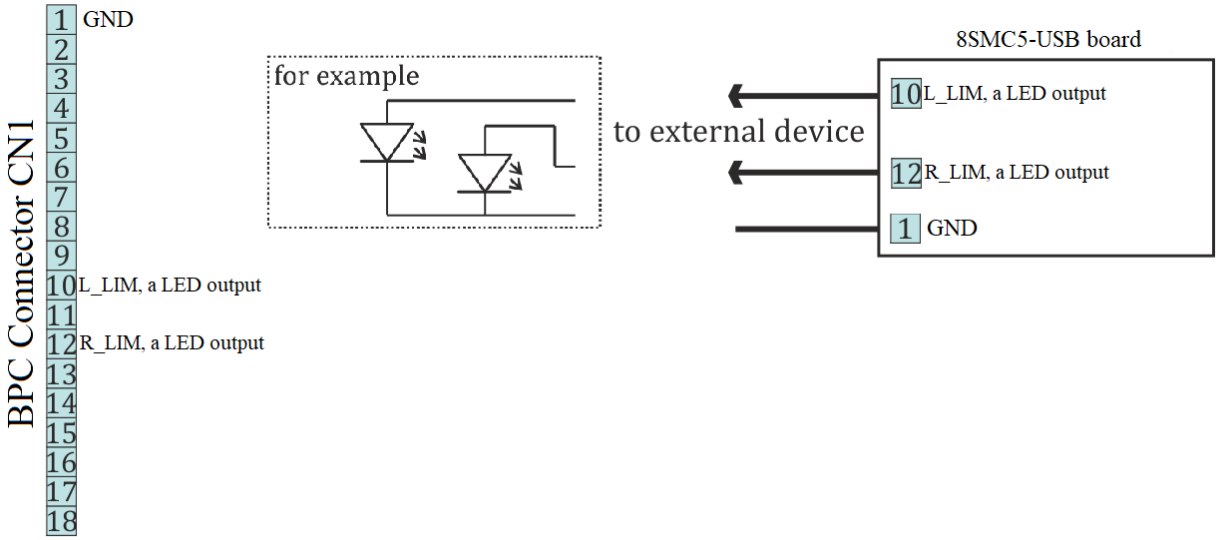


Fig. 4.24: Connection of the limit indicators to the controller board

4.5.1.3.2 One-axis and two-axis systems

LEDs on the front panel of controllers in the box (*one-axis* and *two-axis* systems) are indicators for power, status and limit switches, therefore a connection diagram is not required.

4.5.2 Operations with magnetic brake

There is an output pin on BPC connector for magnetic brake control, which is installed on the stepper motor shaft. Magnetic brake is used hold motor position in unpowered state.

4.5.2.1 Description of operation

Magnetic brake consists of a magnet and a spring, which performs stops the motor shaft. In case there is no voltage applied to the magnet the spring clamps the shaft in place which allows to keep motor position. When voltage is applied the spring releases the shaft.

4.5.2.1.1 Controller operating sequence during stage shutdown.

Engine shutdown (the time of shutdown is recorded in controller) -> magnet power supply cut off, shaft fixation -> board power supply cut off

During power-on the sequence is reversed.

Since any movement has inertia, the following parameters are set to control magnetic brake and the position fixation process:

- Time between motor power-on and brake deactivation (ms)
- Time between brake deactivation and readiness for movement (ms)
- Time between engine stop and brake activation (ms)
- Time between brake activation and power-off (ms)

If magnetic brake function is turned off then controller will constantly transmit brake release signal. This allows to move engine equipped with magnetic brake without rotor fixation during pauses. If winding poweroff function is turned off then controller will only pause between brake switching and movement start/stop.

All magnetic brake settings can be changed online and brake will be switched to the mode which would be active in case if the setting would have a new value. For example a large increase of brake activation delay when the brake is already active will lead to brake deactivation and countdown to the new delay value. It is also possible to turn on or off magnetic brake itself or winding power function.

Table 4.6: Output electric parameters

Type	TTL
Active condition (brake is released)	0.3 V
Passive condition (brake is not powered)	0 V
Operational current	no more than 4 mA

Magnetic brake setting in XIlab program is described in *Brake settings* section.

4.5.2.2 Magnetic brake connection diagram

For operation with the magnetic brake a special board controlled by digital signal is used. One-axis and two-axis systems, which equipped with this board and able to work properly with magnetic brake are available separately (look *below*).

4.5.2.2.1 Controller board

A contact for the magnetic brake control is located on the *BPC connector*.

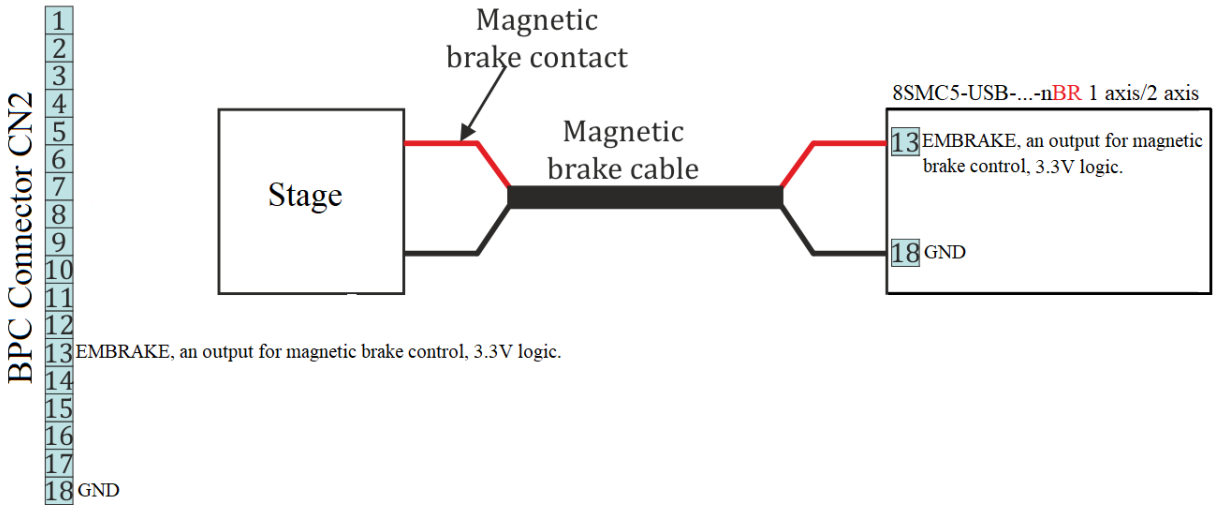


Fig. 4.25: Connection of magnetic brake to the controller board

Power converter is a converter from digital signals to power. If magnetic brake output is high, the magnetic brake of stage has 24V on its power input. If it is low, the magnetic brake has 0V on its power input. In the most common case a scheme with transistor and diode is used. For more detailed information please contact us: 8SMC4@standa.lt.

4.5.2.2.2 One-axis and two-axis systems

In order to use magnetic brake check if your controller system is equipped with a special converter board. Models which meets this demands can be identified by letters **BR** in its title, for example **8SMC5-USB-B8-1BR**.

Contact pin responsible for magnetic brake control in boxed versions of controller is located on the *Supplementary one-axis system connector*. A connection diagram is shown below. *Two-axis* systems are supplied with only one axis which compatible with magnetic brake.

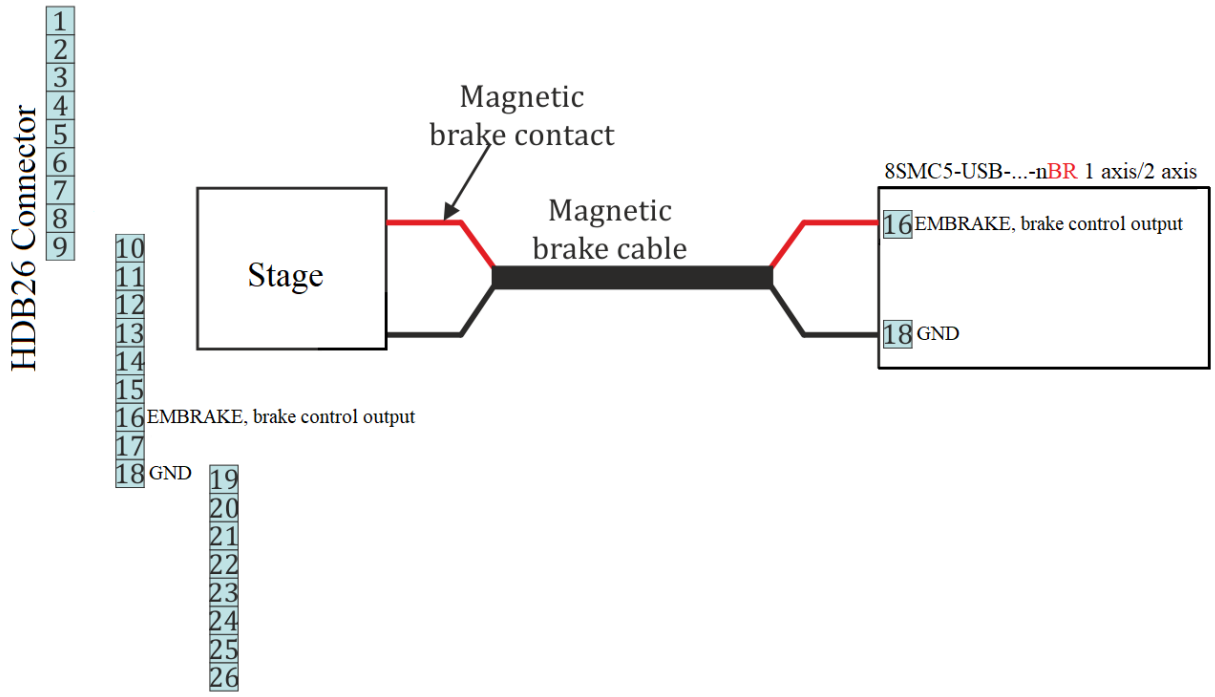


Fig. 4.26: Connection of magnetic brake to one-axis or two-axis systems

4.5.3 Joystick control

4.5.3.1 General information

Controller accepts an input from an analog joystick with voltage in 0-3 V range. Voltage in the equilibrium (central) position and voltage in minimum and maximum position can be set to any value from the working range, if the following condition holds: minimum position < central position < maximum position. Controller uses digital representation of joystick input values: 0 V corresponds to a value of 0 and 3 V corresponds to a value of 10000.

To stop movement in the central position a DeadZone option is available, which is counted from the central position and measured in percent. Any joystick position inside deadzone leads to the stopping of the movement by the controller. A larger than deadzone deviation of the stick starts controller movement with the speed which is calculated from the deviation. One can reverse the joystick with a reverse flag which can be useful to keep “right joystick offset means movement to the right” correspondence for any physical orientation of the joystick and the stage.

Movement speed has an exponential dependence on joystick deviation from the center. This enables one to reach high precision through small joystick shifts and high speed through large ones. Nonlinearity parameter (Exp factor) can be varied. If the nonlinearity parameter is zero, then the motor speed will linearly depend on joystick position.

The following graph shows dependence of movement speed on joystick position for the following settings:

Central deviation	4500
Minimum deviation	500
Maximum deviation	9500
Dead zone	10%
Maximum movement speed	100

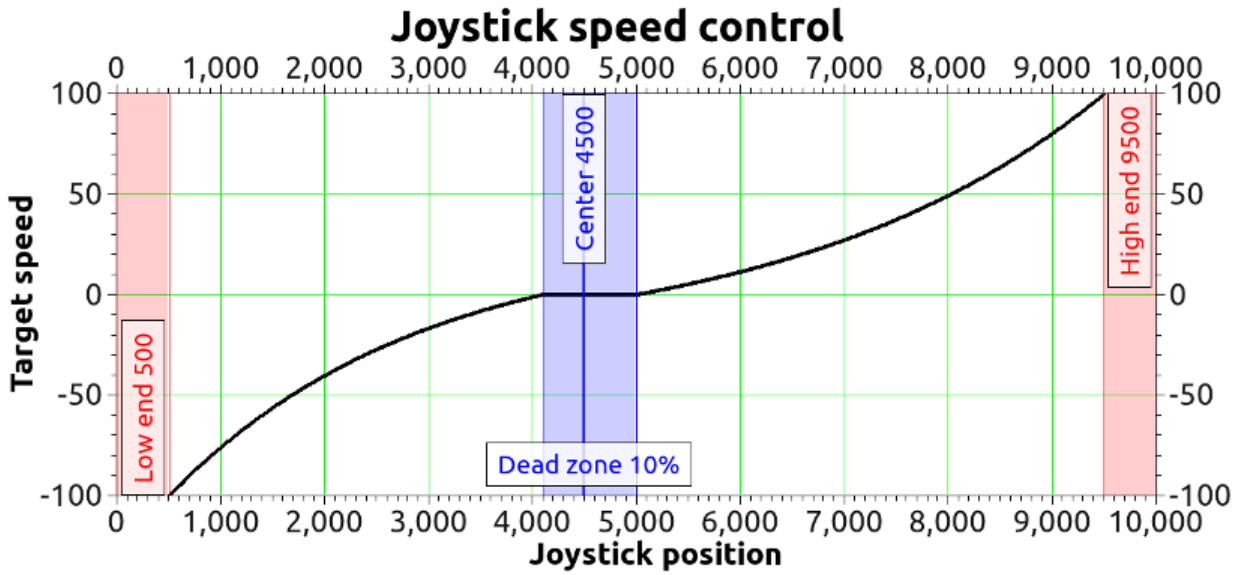


Fig. 4.27: An example of motion speed dependence on joystick deviation

Sometimes exponential joystick response which combines high accuracy and high speed is not enough. That's why controller supports maximum speed table. User can switch between these speeds using *Left-Right buttons control*. In case of right key press maximum speed is changed from *Max speed*[i] to *Max speed*[i+1]. In case of left key press maximum speed is changed from *Max speed*[i+1] to *Max speed*[i]. Controller starts with i=0. There are 10 values in the speed array. If both integer and fractional parts of *Max speed*[x] are equal to zero then it isn't possible to switch to this speed from *Max speed*[x-1]. This is made so that a user can limit speed array size. An attempt to get out of array bounds will also lead to nothing.

Controller has contact debouncing on control buttons. For keypress to register button press should last longer than 3 ms.

If joystick sits within dead zone for more than 5 seconds it will be logically considered to be out of deadzone only when it has been physically out of deadzone for more than 100 ms. This allows user to release joystick and to be confident that even occasional noise on joystick output won't lead to unnecessary motor motion. While joystick is within *Dead zone* the controller can receive any commands from computer including motion commands, home position calibration commands, etc. If during command execution joystick is brought of *Dead zone* the motion command is canceled and motor is switched to joystick control. This allows the user to turn on joystick control mode and use it only when necessary.

Everything that is related to movement under the control of controller commands is also applicable to joystick movement. This includes acceleration, maximum speed limit, windings poweroff delay, magnetic brake, backlash compensation, etc. For example, if you suddenly release joystick handle and let it return into the deadzone, then, if corresponding modes are on, controller will gradually slow the motor, make a backlash compensation motion, stop the motor, fix the motor shaft with the magnetic brake, smoothly reduce current and switch off windings power.

MaxSpeed[i] and DeadZone parameter change is described in *Settings of external control devices*.

Important: In Joystick control mode, the physical and virtual buttons remain in working order

Warning: Do not disconnect or connect the joystick to the switched on controller! When disconnecting/connecting the joystick to the enabled controller, the joystick or controller will not burn, but the stages con-

nected to the controller will start moving to the limit switches.

4.5.3.2 Connection diagram

Important: Analog inputs for joystick connection are designed for a range of 0-3V. Be careful and do not exceed voltage for joystick contacts.

4.5.3.2.1 Controller board

A contact for joystick control is located on the *BPC connector*.

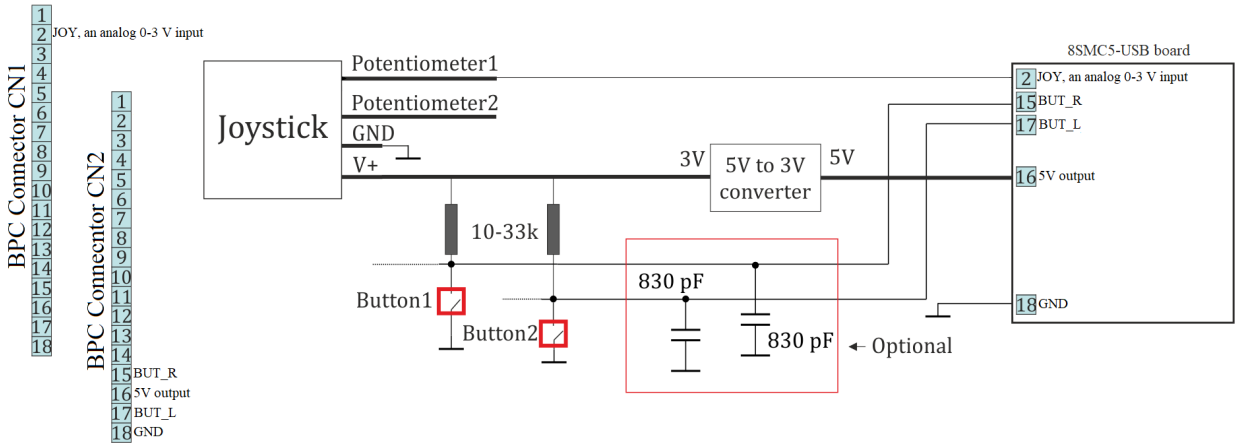


Fig. 4.28: Connection of joystick to the controller board via BPC connector

4.5.3.2.2 One-axis and two-axis systems

Joystick connector is available only in *two-axis system*. A connection diagram is shown below.

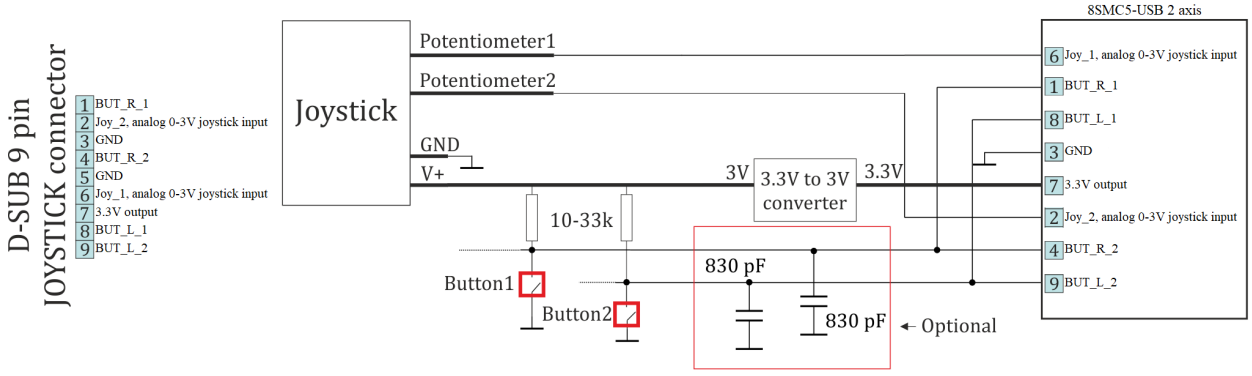


Fig. 4.29: Connection of joystick to the two-axis system via D-SUB 9 pin connector

4.5.4 Left-Right buttons control

For each system it is possible to control the movement of a motor with the buttons. Active button state is programmable and can be logical zero or one. Controller supports a 10-item speed array MaxSpeed[0-9], which is used both for *joystick* and button control.

The buttons control settings are red/written by commands SCTL/GCTL (*set_control_settings*/*get_control_settings*).

- If a left or right button is clicked then motor does a shift on an offset, specified by DeltaPosition and uDeltaPosition.
- If a left or right button is pressed and held for longer than MaxClickTime, then motor starts moving with MaxSpeed[0] and counting down to Timeout[0]. After Timeout[i] microseconds have elapsed speed is changed from MaxSpeed[i] to MaxSpeed[i+1] for any i between 0 and 9 (inclusive).
- If press the two buttons, the controller performs a *stop with deceleration*. Holding both buttons for 3 seconds starts the *automatic calibration of the "home" position*.

Note: You can fill only the upper part of the 10-item speed array if you don't need all of them. Controller changes its speed to the next one only if the target speed is not zero and the timeout is not zero. For example, if MaxSpeed [0] and MaxSpeed[1] are nonzero and MaxSpeed[2] is zero (both step and microstep part), then the controller will start moving with MaxSpeed[0], then change its speed to MaxSpeed[1] after Timeout[0] and will keep moving with MaxSpeed[1] until the button is released. You can also set Timeout[1] to zero and leave MaxSpeed[2] set to any value to achieve the same result. Controller obeys its movement settings (with the exception of target speed). For example, when changing its speed from MaxSpeed[i] to MaxSpeed[i+i] controller will either accelerate with set acceleration value or change its speed instantly if acceleration is disabled.

The default state is set according to the voltage levels of the buttons (*Output parameters*). The state of each button can be software inverted. When active, the button is considered to be down. It does not matter how the condition is active (after changing the invert states, or when changing the voltage level at the physical impact of a button). The controller uses button contact debouncing. The button is considered pressed if active state lasts for longer than 3 ms.

Table 4.7: Output parameters

Type	TTL
Logic zero level	0 V
Logic one level	3.3 V

Warning: When you turn on or reboot the controller at the input voltage level of the button is present, which is considered to be active, the controller will accept it as a button is pressed, and begin to obey the *rules described above*.

4.5.4.1 Connection diagram

4.5.4.1.1 Controller board

"Right" and "Left" control buttons can be connected to the controller board via *BPC connector* for a motor motion control. There are no integrated pull-down resistors for setting of input potential by default, that's why it is necessary to use (pull up) and (pull down) resistors scheme.

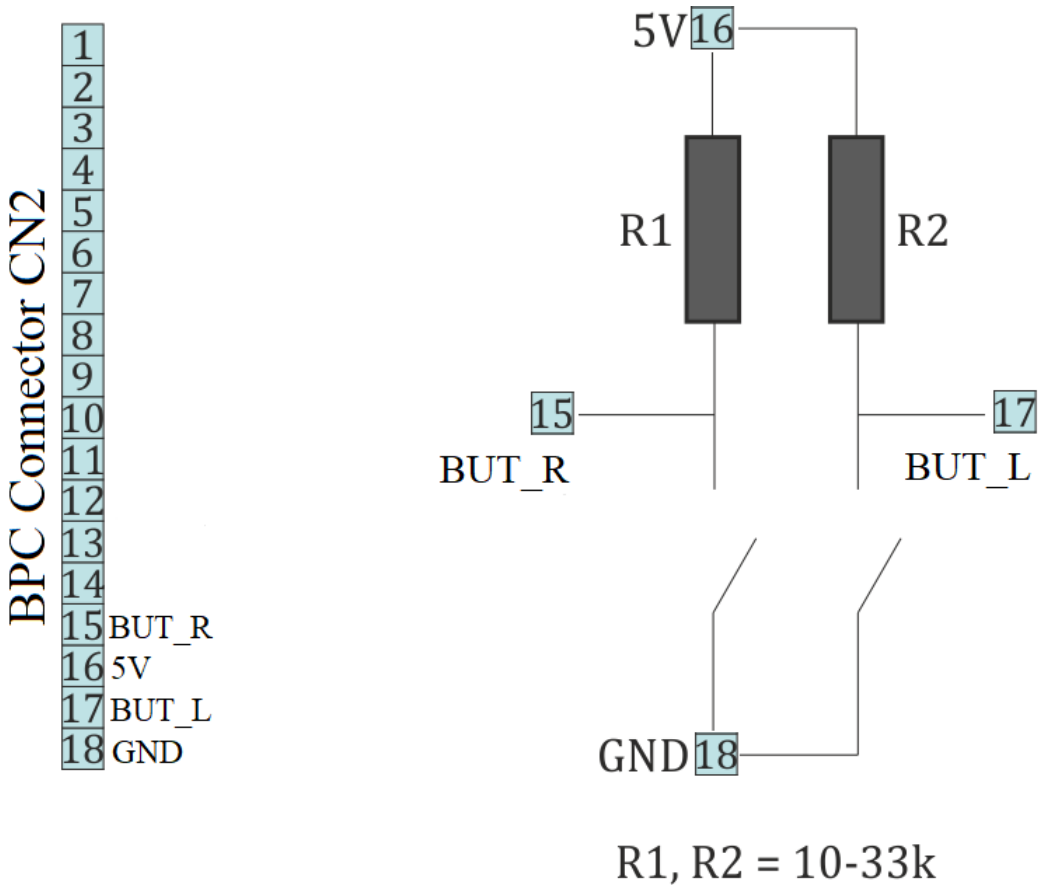


Fig. 4.30: Scheme of buttons connection to the BPC connector

4.5.4.1.2 One-axis and two-axis systems

For controllers in case user buttons already placed on the front panel. However it is possible to connect custom buttons to the corresponding contacts that located on the *D-SUB 9 pin connector* and available only in *two-axis* system. Connection diagram is shown below.

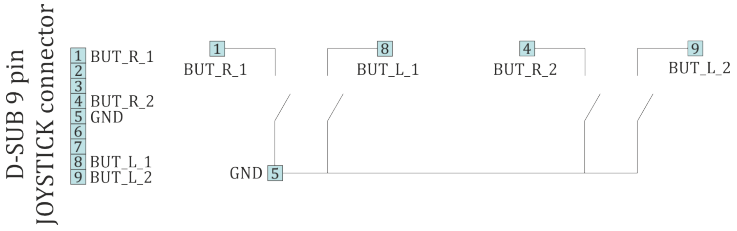


Fig. 4.31: Scheme of buttons connection to the HDB-26 connector

4.5.5 TTL synchronization

4.5.5.1 Principle of operation

TTL-synchronization is used to synchronize controller motion with external devices and/or events. For example, the controller can output synchronization pulse each time it moves a certain distance. Vice versa, controller can shift a certain distance on incoming synchronization pulse, for example from an experimental setup which is ready to move

to the next measurement position.

To use mechanical contacts as an input synchronization signal a contact debouncing is provided. One can set minimum input pulse length which is recognized as a valid synchronization signal. An active state is a logical one (see *Input parameters*), and a raising edge is considered to be the start of a signal. However, if for some reason this is undesirable, both options may be inverted independently.

Table 4.8: Input parameters

Type	TTL
Logic zero level	0 V
Logic one level	3.3 V

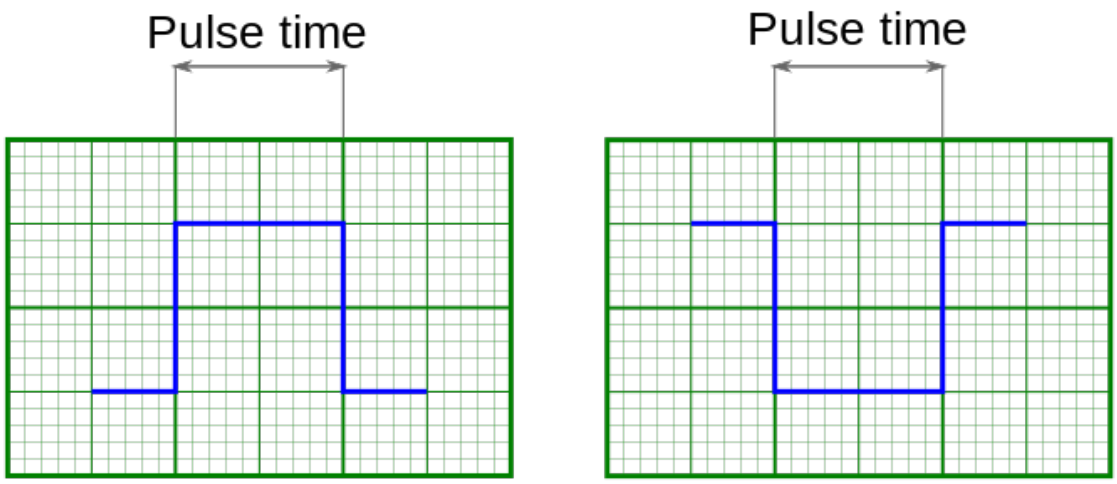


Fig. 4.32: Inversion of input and output syncronization pulse illustrated

Note: If simultaneous start of several controllers in a multiaxis system is desirable, minimum input pulse length should be the same for all controllers. Contact debouncing should not be used in systems with no mechanical contacts and short noise pulses in synchronization input channel. One should use an RC-circuit which would filter these noise pulses instead.

Synchronization in important in multiaxis systems because it allows one to start movement on several axes simultaneously. To do this all axes are prepared to start the movement, all slave axes are set to start moving on input synchronization pulse, one master axis is set to output a synchronization pulse on the start of the movement. Master axis output is connected to slave axes' input. In this setup any movement of the master axis leads to immediate response of all connected axes.

Note: One should set minimum input sync pulse length to 0 if this kind of connection is used. This disables contact debouncing, but since there are no mechanical contacts it is not needed. If minimum input sync pulse length is not zero then to avoid desynchronization of master and slave axes one should set input sync pulse length the same for all controllers, connect syncout to both master and slave inputs and issue start command by activating input manually.

Synchronization input and output are independent from each other and other means of motion control. Control through *XiLab application* or any other user application, *joystick control* and *left-right buttons control* are independent of

input/output synchronization state. Last command always takes priority. For example, a movement command sent from XiLab will cancel current movement which happened because of input sync pulse, but will not affect output sync state. Next input sync pulse will cancel current movement initiated by user program and will replace it with movement command according to sync in settings.

Note: Sync in settings may be saved in controller flash (non-volatile) memory. In this case everything related to synchronization may also be said about autonomous controller operation. For example, you may set up shift on offset on syncin pulse with syncout pulse on movement stop and connect the controller to a standalone measurement device, which starts measurements on its own input sync pulse and outputs a sync pulse on measurement end. Then you can run such a system without a PC, because after the first sync pulse all measurements and movements will happen automatically.

4.5.5.2 Connection

The controller is supplied with two TTL-sync channels on the *BPC connector*.

4.5.5.3 Sync in

Syncronization input has a setting, which defines minimum syncin pulse length which may be registered. This length is measured in microseconds. Use this setting to decrease controller sensitivity to noise. Synchronization input may be turned on or off. If it is on, then a sync in pulse will lead to a situation as if *Predefined displacement mode* command has taken place, which takes its *Position* and *Speed* from syncin settings. If syncin settings are changed during the time the movement takes place it will not change current movement parameters. Movement parameters will change on the next front on syncronization input. This designed deliberately to allow one to set up next shift parameters in multiaxis systems during movement.

Warning: When you turn on or reboot the controller at the input voltage level of the synchronize input is present, which is considered to be active, the controller interprets it as if *Predefined displacement mode* command has taken place.

Note: *Position* and *Speed* are two separate variables which also may be saved in non-volatilve controller memory. They are used only with synchronization input.

Note: Syncin movement obeys acceleration, max speed settings and all other settings which are related to motion. Their incorrect setting may disrupt coordinated movement in multiaxis system.

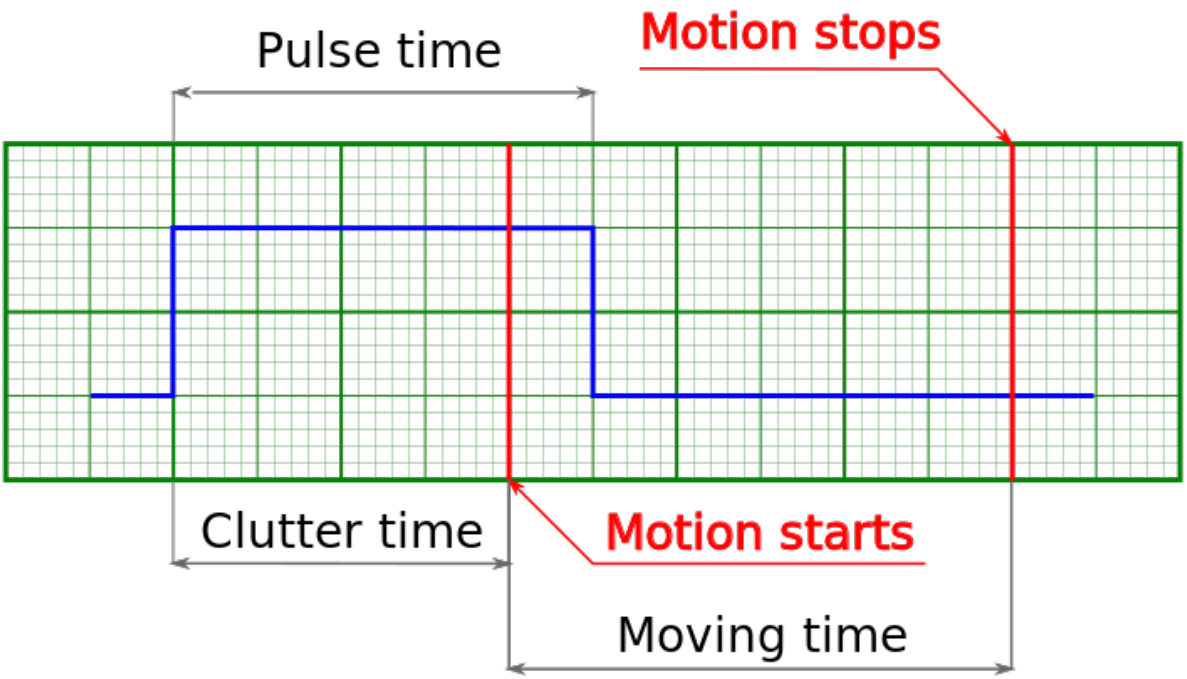


Fig. 4.33: Movement starts becase input pulse is longer than debounce time

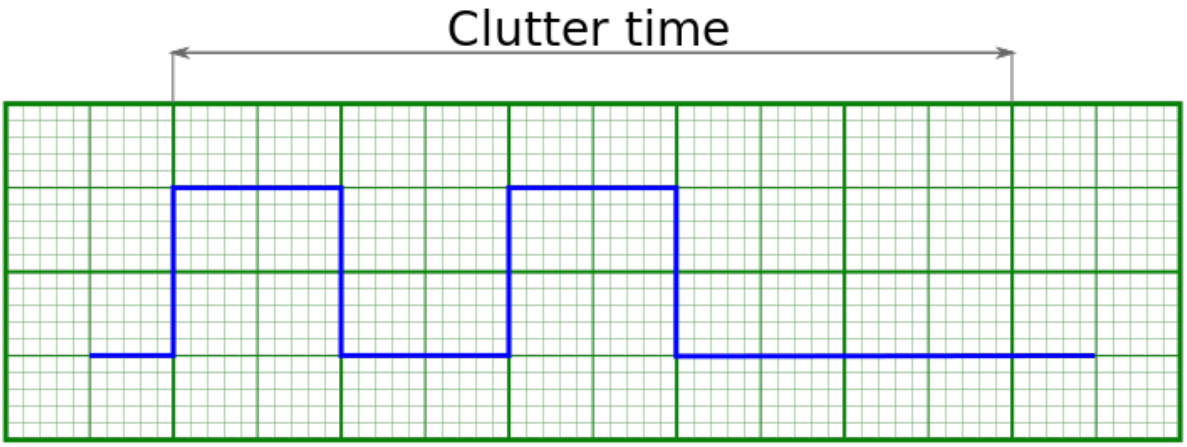


Fig. 4.34: Movement does not start because input pulses are shorter than debounce time

Warning: If a second syncin pulse is received while controller is still moving then the end position will be offset by two times the shift distance from the start, if a third pulse is received, then by three times, etc.

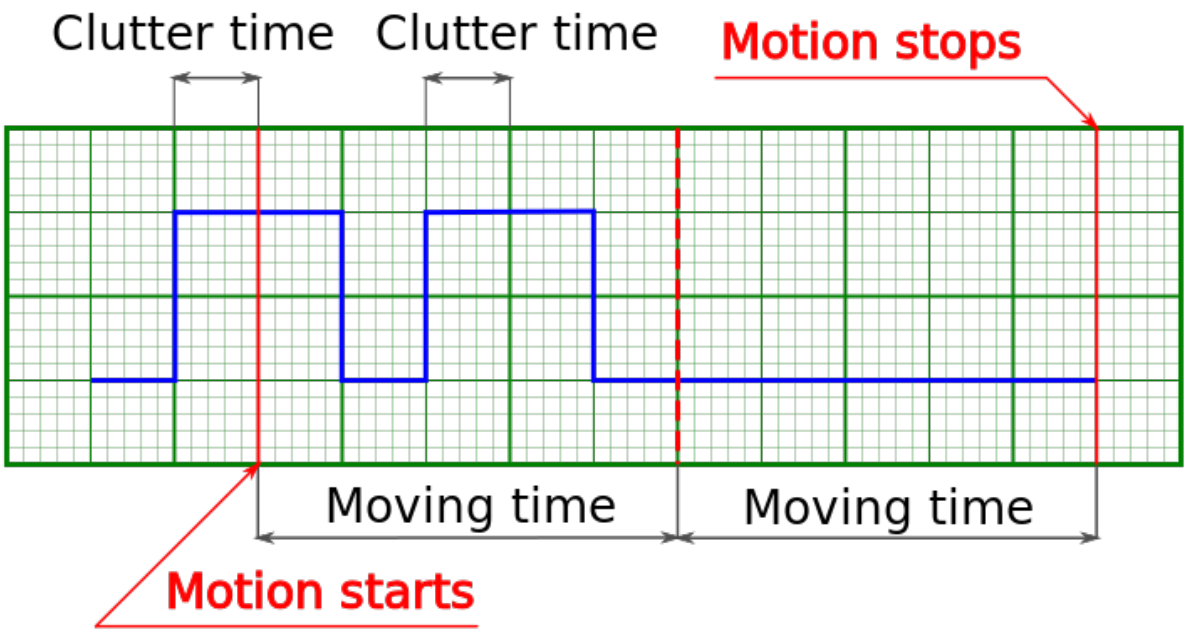


Fig. 4.35: One-time movement with double shift length because second syncin pulse came in before the end of the movement

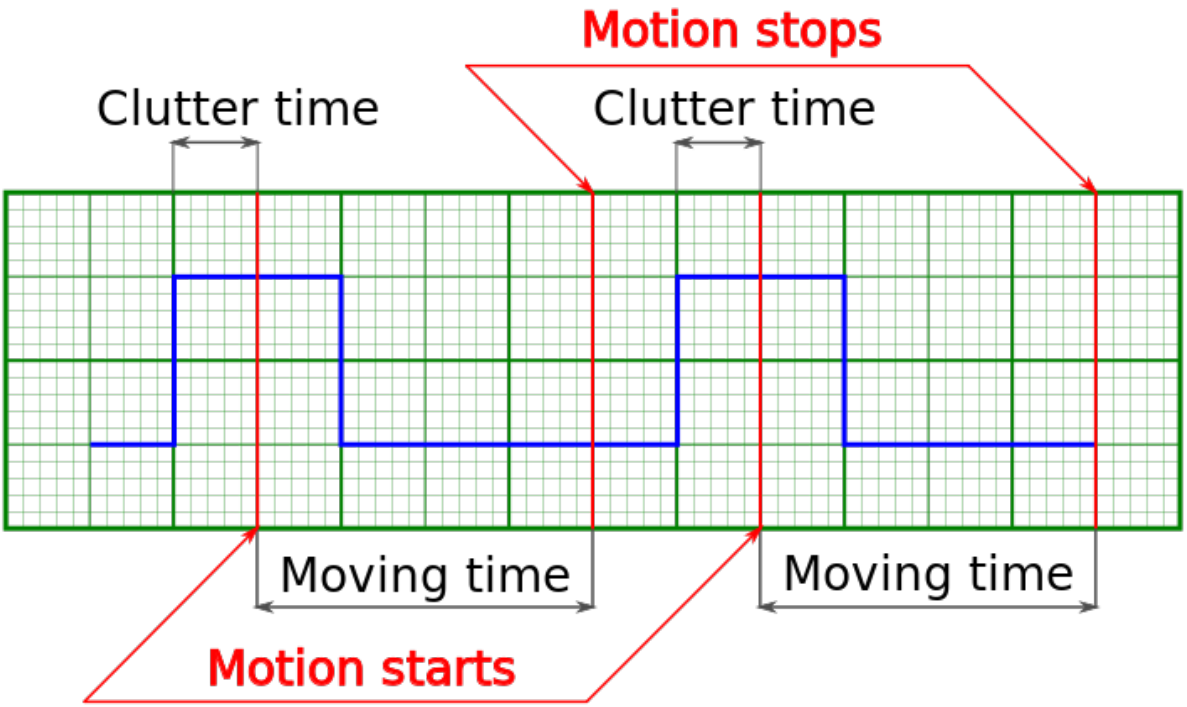


Fig. 4.36: Two separate shifts with two start and two stop phases

Default setting is active state is one, movement on raising edge. Synchronization input may be inverted to the active state is zero, movement on trailing edge.

Note: Inverted synchronization input setting will lead to the change in the definition of active/inactive state which may be seen, for example, in *controller status*. However, programmatical inversion of the syncin state by itself will not lead to the start of the movement, even if the transition happened into the active state.

4.5.5.4 Sync out

Output synchronization is used to control external devices tied to controller movement events. Sync out pulse can be emitted on start and/or stop of the movement, and/or on each shift on the preset distance. *ImpulseTime* setting defines the length of sync pulse, either in microseconds or in distance units. Synchronization output can be switched into general purpose digital output mode. In this mode it is possible to set zero/one output logic level programmatically.

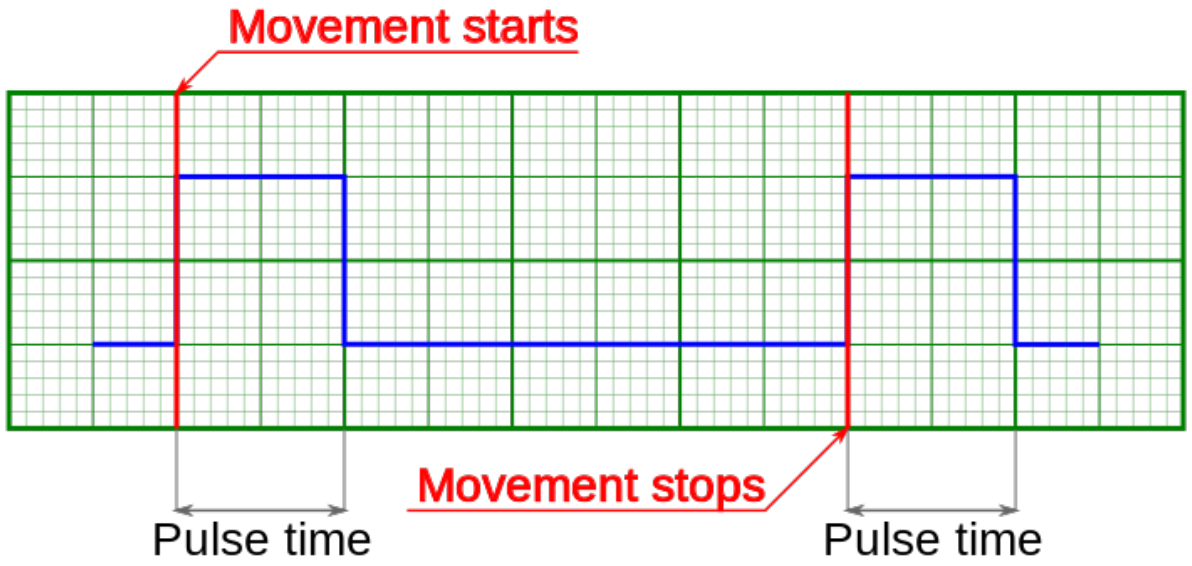


Fig. 4.37: Sync out pulses generated on start and stop of the movement (fixed length pulse)

Note: If syncout pulse length is measured in distance units and, for example, is equal to 10 stepper motor steps and "syncout pulse on stop" mode is active, then the active state on synchronization output will be set on the movement end, but will be cleared only when the motor will move 10 more steps during the next movement.

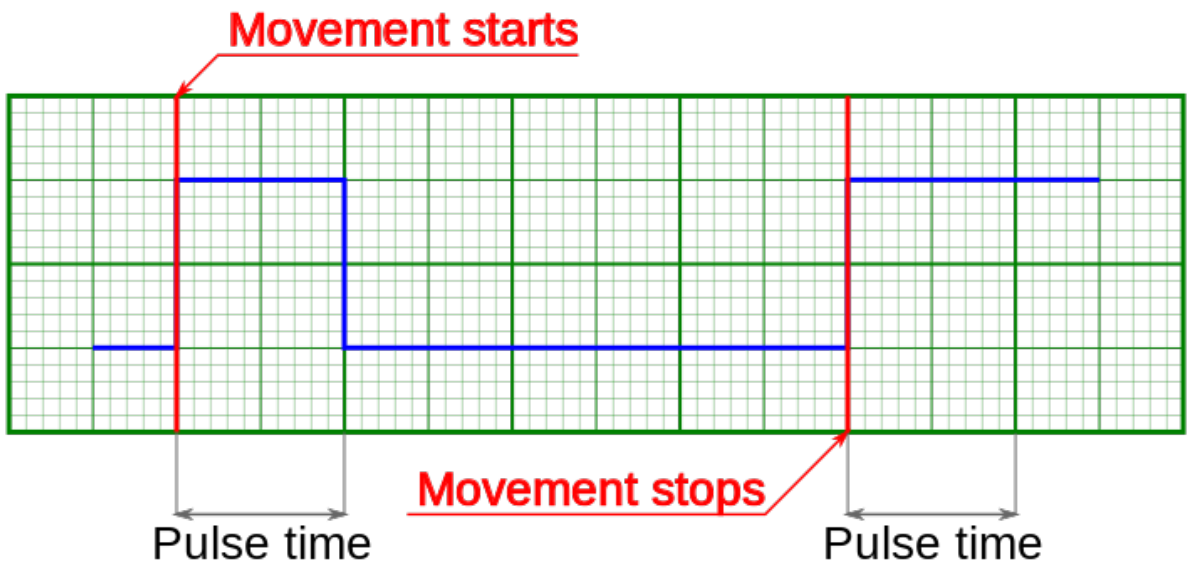


Fig. 4.38: Syncout pulses generated on start and stop of the movement (pulse is measured in distance units)

Note: If you wish to reconfigure synchronization output and are not sure which state is it in, change its state to general purpose output and set the desired logic level.

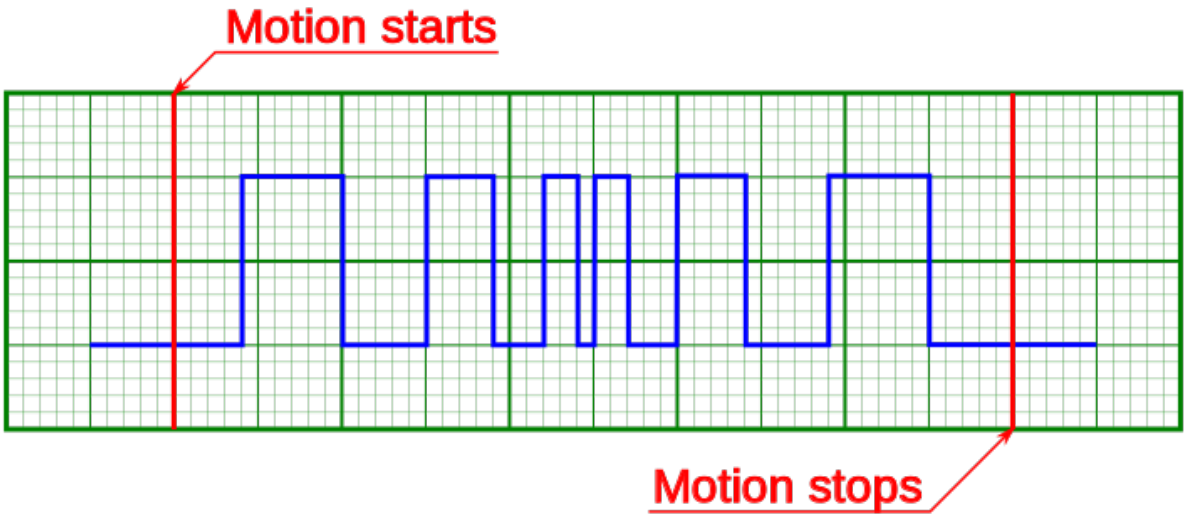


Fig. 4.39: Sync out pulses on movement with acceleration and "generate on shift" mode (pulse length measured in distance units)

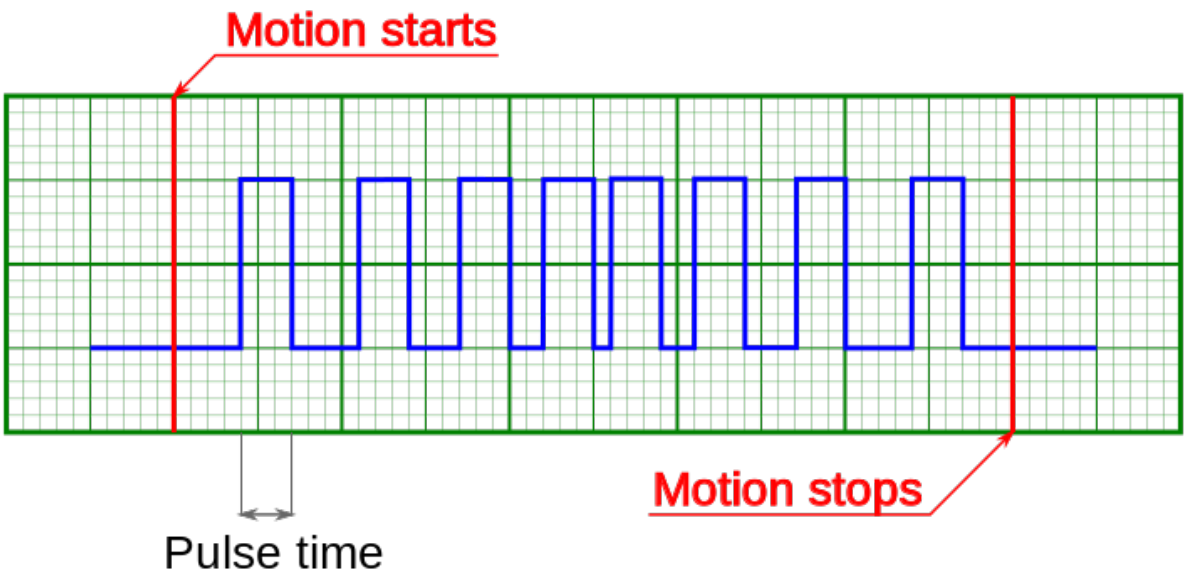


Fig. 4.40: Sync out pulses on movement with acceleration and "generate on shift" mode (pulse length measured in microseconds)

Note: Periodic syncout pulse generation imitates revolution sensor with reducing gear. Coordinates which trigger syncout pulse generation are counted from zero position and not from the position the controller is in at the start of the movement. For example, if synchronization output is set up to generate pulses every 1000 steps then pulses will be generated in positions 0, 1000, 2000, 3000, etc. Pulse generation works when moving in both directions. Pulse is generated when the quotient of current coordinate and pulse generation period changes. That is, pulse is generated when position 1000 is reached when moving in the direction of increasing position and it is generated when position 1000 is left when moving in the direction of decreasing position. Also, syncout pulse is always generated when position 0 is reached from any direction (including the case when position is reset by the ZERO button).

Note: Whenever syncout pulses overlap they are merged into one pulse.

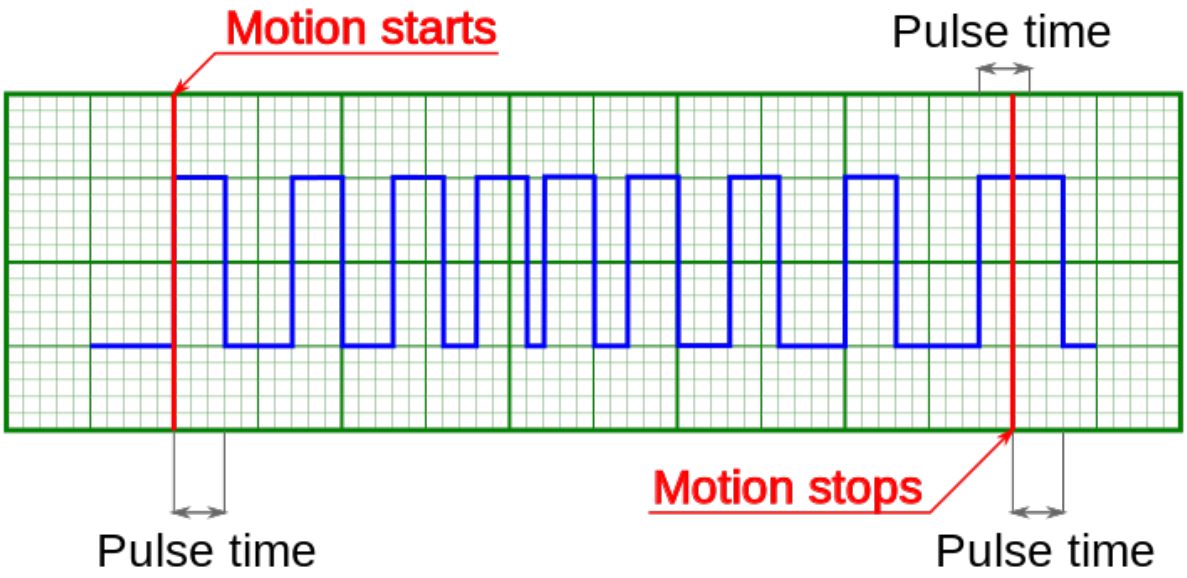


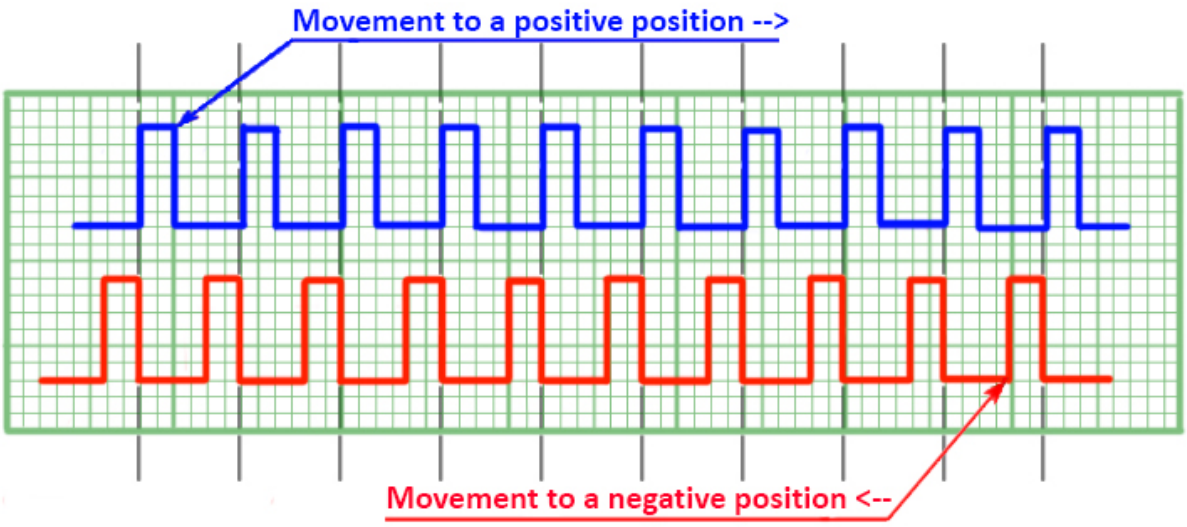
Fig. 4.41: Syncout pulse merge illustrated, pulse generation on start, stop and shift on offset (pulse length measured in microseconds)

The controller will set virtual marks with a specified step that corresponds to the value of the "Every" field starting from zero. The synchronization pulse is always generated after passing the next mark. Therefore, the position of the pulses depends on the direction of movement:

- When moving in a positive direction (position increases, pulses are generated in the direction of movement), that is, they grade the position of the mark
- When moving in a negative direction (position decreases, pulses are generated in the direction of movement), that is, they are smaller than the position of the mark

Example: Pulse width: 100 Every flag: 1000 The controller will set virtual marks: . . . , -2000, 1000, 0, 1000, 2000, . . .

- When moving from -1500 to 1500, the output will be a logical unit when passing the following coordinate ranges: [-1000, -900], [0, 100], [1000, 1100]
- When moving in the opposite direction from 1500 to -1500 logical unit when passing the following coordinate ranges: [1000, 900], [0, -100], [-1000, -1100]



Important: With short movements within the pulse duration around the mark, the output state may not return to zero, so as not to create unnecessary noise switching. The “Every” flag was not designed for single shifts, it was created to generate pulses over long distances

Synchronization settings setup in XiLab is described in *Synchronization settings* section.

4.5.5.5 Connection diagram

4.5.5.5.1 Controller board

Controller board contains two TTL-channels of synchronization on the *BPC connector*.

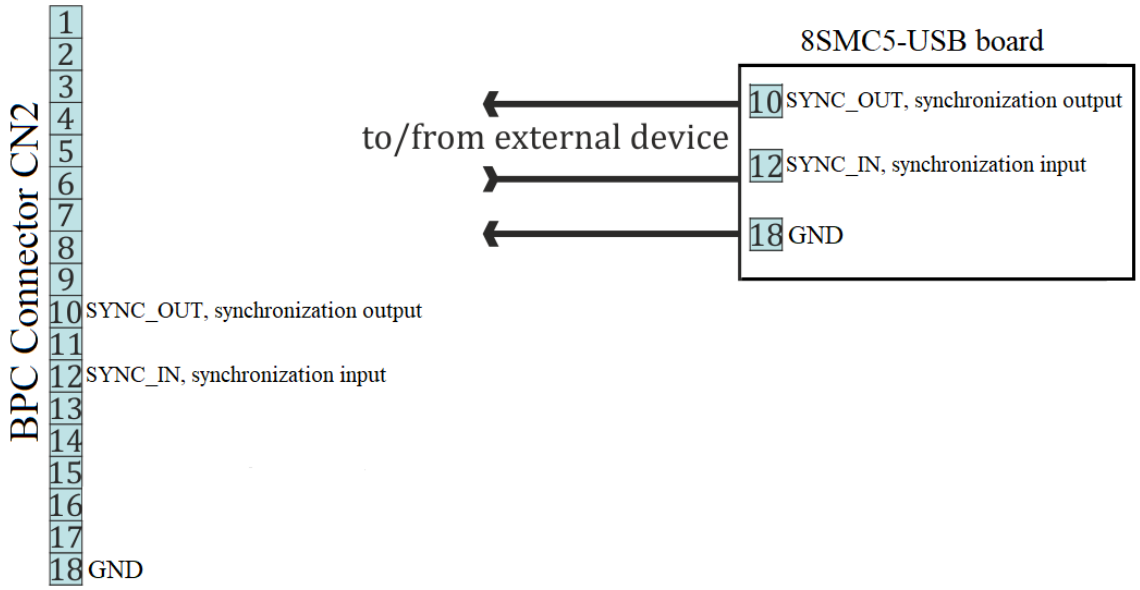


Fig. 4.42: Scheme of connection to the synchronization channels for the controller board

4.5.5.5.2 One-axis and two-axis systems

Synchronization signals on the *one-axis* and *two-axis* systems are located on the *HDB-26 connector*.

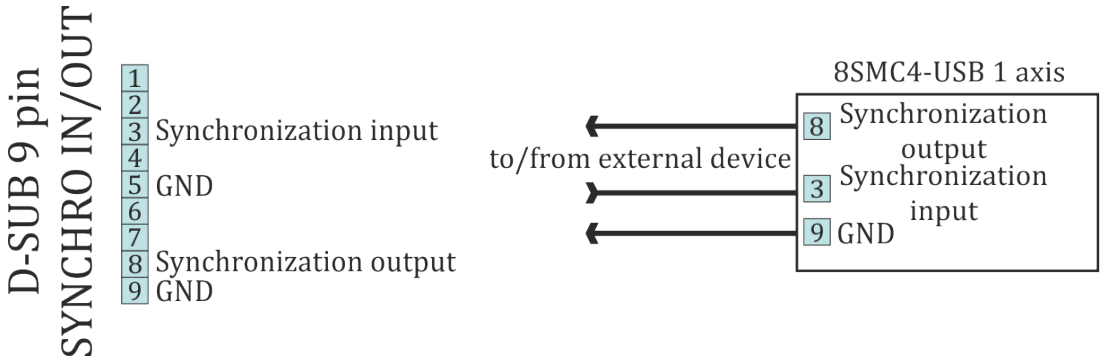


Fig. 4.43: Scheme of connection to the synchronization channels for the one-axis system

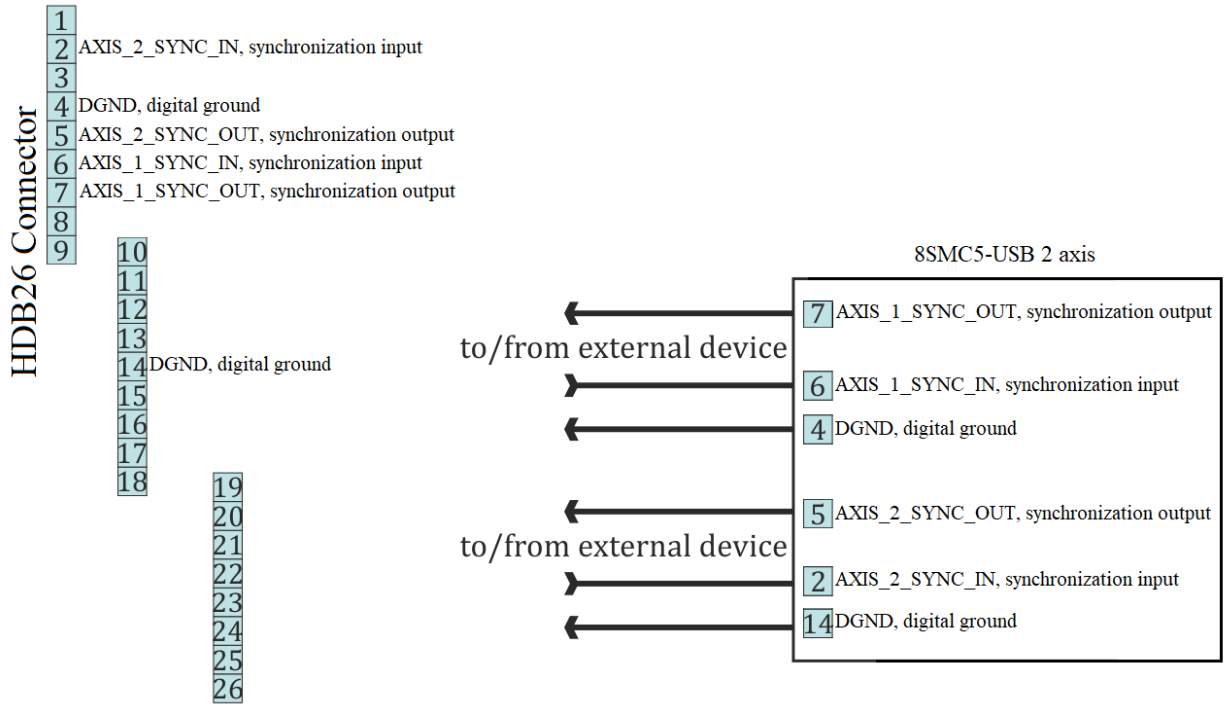


Fig. 4.44: Scheme of connection to the synchronization channels for the two-axis system

4.5.6 Multiaxis system design

Controller axes in multiaxis systems are identified by the controller serial number. Each controller has its own unique serial number, which may be seen in XiLab software on *About controller* page. One can read controller serial number using **get_serial_number** function (see *Programming guide*).

Multiaxis systems based on this controller are assembled using an active backplane based on a USB-hub or an external USB-hub.

Important: It is recommended to base backplanes on a USB-hub with galvanic isolation from the PC and an additional power supply or a 36 V -> 5 V converter, because this provides higher noise immunity and guarantees sufficient 5 V power.

For proper multiaxis system functioning one should first connect all controllers to the power and USB connectors (place controllers on the backplane).

Then, in any order, do all of the following:

- Connect power to the backplane
- Connect external devices
- Connect master controller to the USB

Note: Note. Additional 5 V power supply requirements: output current no less than 250 mA per axis, 400 mA per axis for full functionality.

Multiaxis connection diagram implies connecting power supply to pins 1 and 2 of the BPC connector and connecting data lines D-/D+ from the USB hub to pins 3 and 4 of the BPC connector.

Power is connected next:

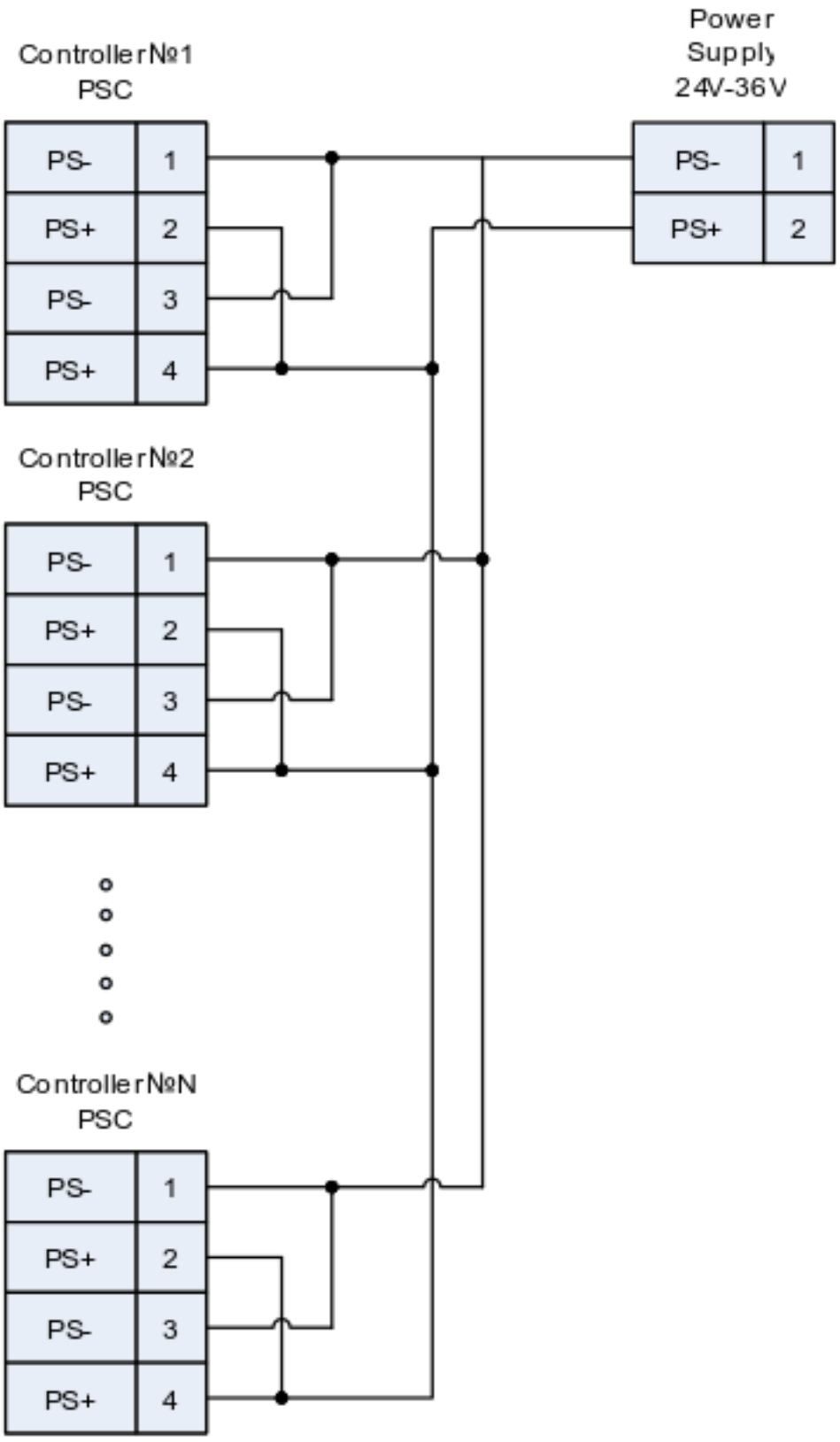


Fig. 4.45: Controller connection diagram in multiaxis configuration

PSC - Power Supply Connector, used to connect power supply to the controller

BPC - Back Panel Connector, used to connect accessories to the controller

4.5.7 General purpose digital input-output (EXTIO)

Output is located on *BPC connector*. It allows user to configure it as input or output. Logical level one is considered to be active (see *Input parameters* table). However it can be inverted so that logical level zero is considered active.

Table 4.9: Input parameters

Type	TTL
Logic zero level	0 V
Logic one level	3.3 V

In input mode you can get information about logical level on input (see *Controller status*), or initiate the following actions during transfer to active state (or during transfer to non-active state if the input is inverted):

- Perform *Command STOP* (quick stop).
- Perform *Command PWOF* command (windings power supply switch off).
- Perform *Command MOVR* command (shift to the given distance with last used settings).
- Perform *Command HOME* command (automatic position calibration).
- Enter *Alarm* state (turn off H-bridges and wait reinitialization).

It does not matter how the state of the input becomes active (after changing the invert states, or when changing the voltage level). The controller uses a software debounce the input. Initiating the action takes place only when the active state of the input buttons lasted for more than 3 ms.

Warning: When you turn on or reboot the controller at the input voltage level of the input is present, which is considered to be active, the controller interprets it as a signal to trigger any of the actions.

Note: Digital input has weak pull down to the ground.

In output mode it is possible to set active or inactive logic level on the following events:

- EXTIO_SETUP_MODE_OUT_MOVING – Active state during motor movement.
- EXTIO_SETUP_MODE_OUT_ALARM – Active state when controller is in Alarm state.
- EXTIO_SETUP_MODE_OUT_MOTOR_ON – Active state while power is supplied to the motor windings.
- EXTIO_SETUP_MODE_OUT_MOTOR_FOUND – Active state while motor is connected.

Table 4.10: Output technical characteristic

Logic type	TTL 3.3V
Update frequency	1 kHz
Nominal current	5 mA

4.5.7.1 Connection diagram

4.5.7.1.1 Controller board

Digital output is located on the *BPC connector*

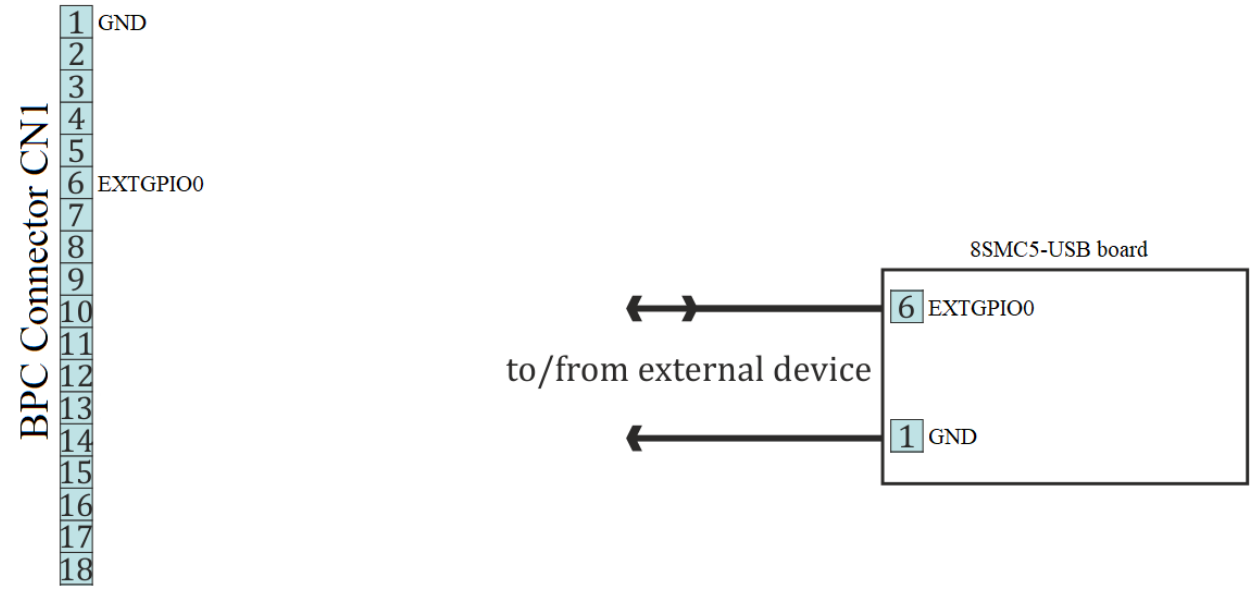


Fig. 4.46: Scheme of connection to digital input/output for the controller board

4.5.7.1.2 One-axis and two-axis systems

Among two box versions, only two axis system has the digital input/output. Corresponding contacts output on the *HDB-26 connector*.

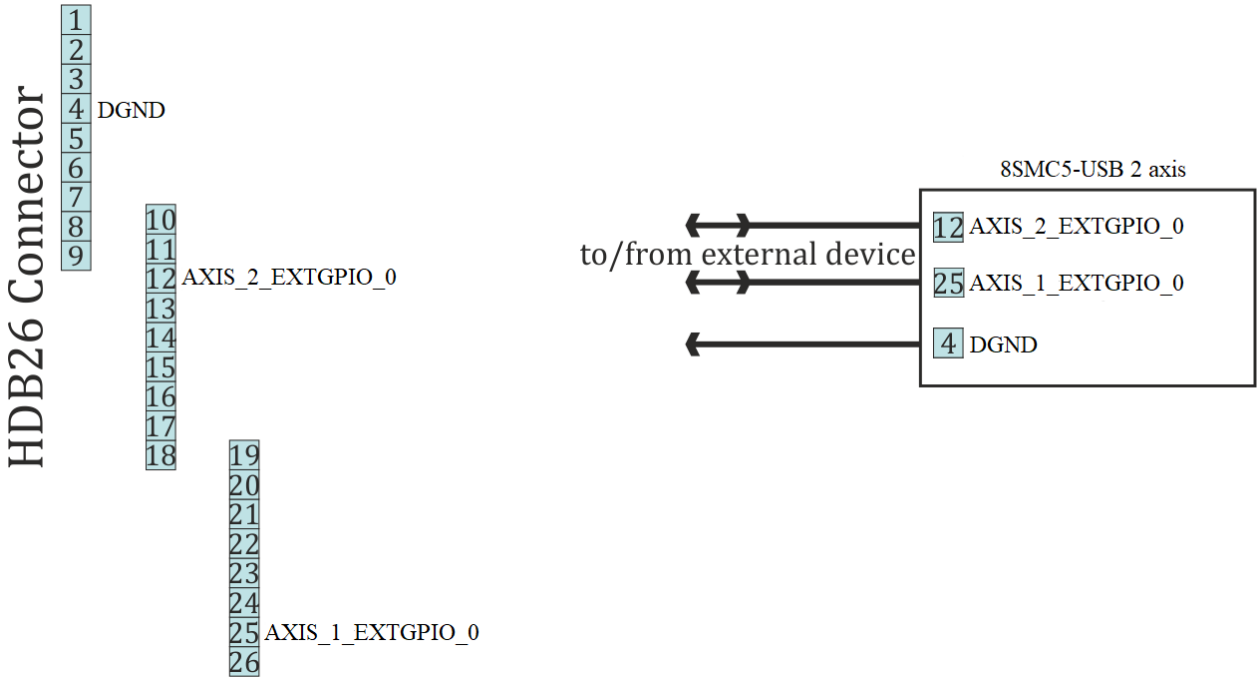


Fig. 4.47: Scheme of connection to digital input/output for two-axis system

4.5.8 General purpose analog input

Analog input may be used for other purpose. For example, it can be used to measure any external signal. Value at the analog input may be read by the *GETC* command and is visible in the *XiLab charts*.

This controller represents analog input values as a number in 0..10000 range. Analog input pin is located on *BPC connector*.

Important: Analog input voltage should not go outside of 0-3 V range. If this voltage is exceeded errors in analog input and other controller subsystems are possible! This may also damage the controller or connected motor.

Table 4.11: Input parameters.

Signal voltage	0-3 V
Scanning frequency	1 kHz

4.5.8.1 Connection diagram

4.5.8.1.1 Controller board

For the *controller board* analog input contact is located on the *BPC connector*.

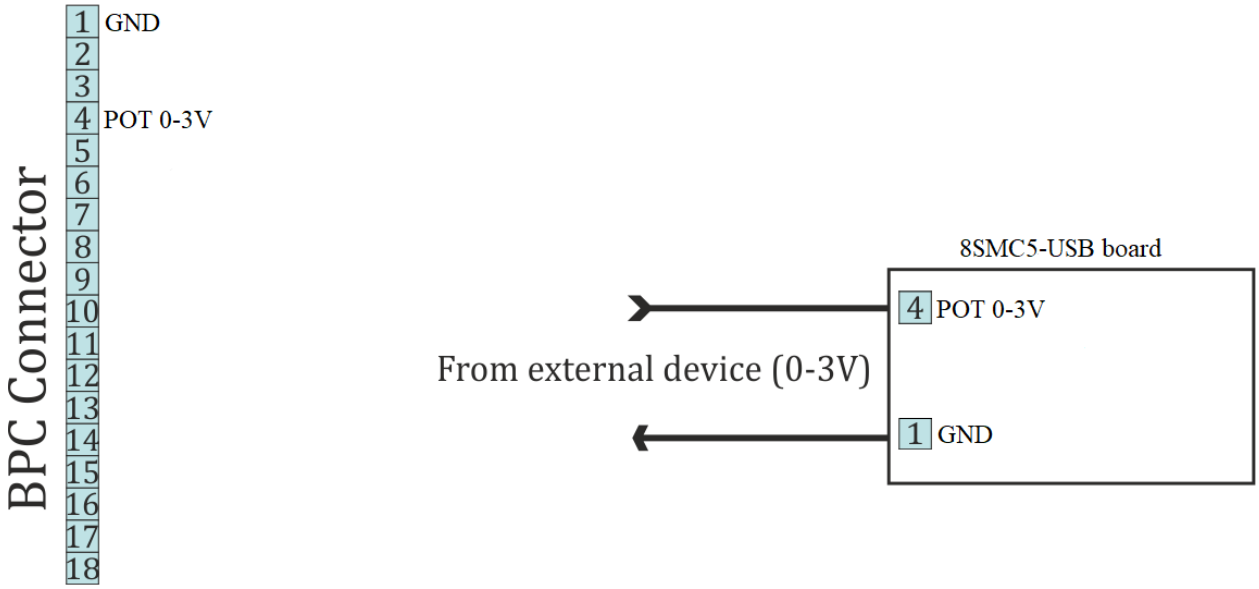


Fig. 4.48: Scheme of connection to analog input for the controller board

4.5.8.1.2 One-axis and two-axis systems

Only *two-axis* system has general purpose analog input on supplementary *HDB-26 connector*.

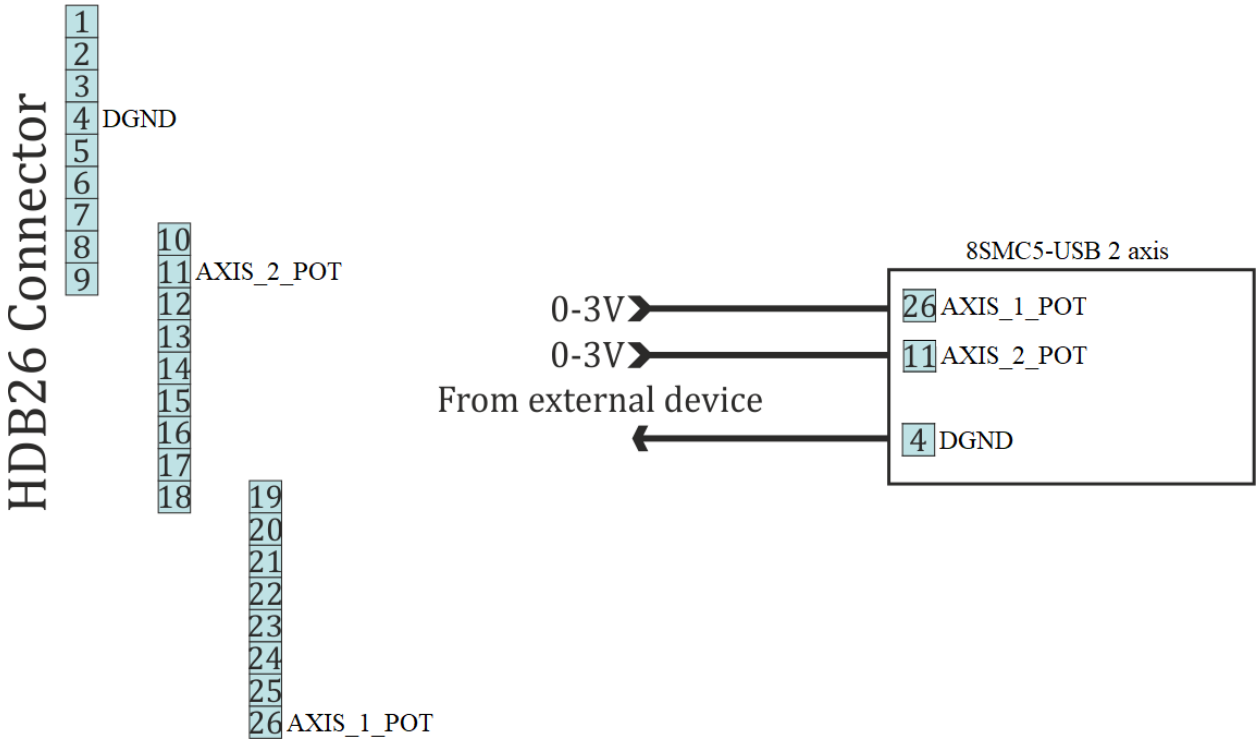


Fig. 4.49: Scheme of connection to analog input for the two-axis system

4.5.9 External driver control interface

Interface allows to control any external driver with a help of 3 standard signals: enable, direction, clock. This mode is convenient when controller power capability is not enough but it is desirable to use its capabilities such as limit switches, revolution sensor, position control, scripting language, multiaxis systems, joystick/button control, magnetic brake, etc. For example, for creation of multiaxis system with one powerful lifting axis which is controlled by external controller and two less powerful horizontal axes, you can use XiLab with 3-axis interface and its scripts and also synchronize the motion of all three axes. I.e. external driver replaces only power part of the controller.

Important: External driver mode works only for **stepper motors** and only in **"None" mode** (xilab settings/Stepper motor tab/Feedback None) In any other modes or with any other type of motor, the External driver mode will not work!

Clock signal defines the quantity of signals in the given Direction (logical one to the right, logical zero to the left). Displacement is a minimum step in the current settings of step division. For step division 1/32 there will be 32 impulses per one step. Don't forget to set external driver in such a way that it would use the same step division.

Warning: Clock signal frequency in the given controller is limited by 78 kHz. That's why to reach necessary speed one might need to reduce step division. Foe example, if rotation speed of 4000 steps per seconds is needed it is necessary to use 1/8 step division or less.

Table 4.12: Output parameters of external driver control

Type	TTL
Logical zero level	0 V
Logical one level	3.3 V

4.5.9.1 Connection diagram

Warning: Outputs for external driver control are not sufficiently protected and can be damaged in case of incorrect usage. Proper connection and necessary electric protections are a responsibility of an engineer who designs drivers' connection system.

4.5.9.1.1 Controller board

For external driver connection three outputs in *BPC connector* are used.

Warning: Pin 13 is the general purpose input/output (see *General purpose digital input-output (EXTIO)*), but it loses its functionality when external driver control is enabled.

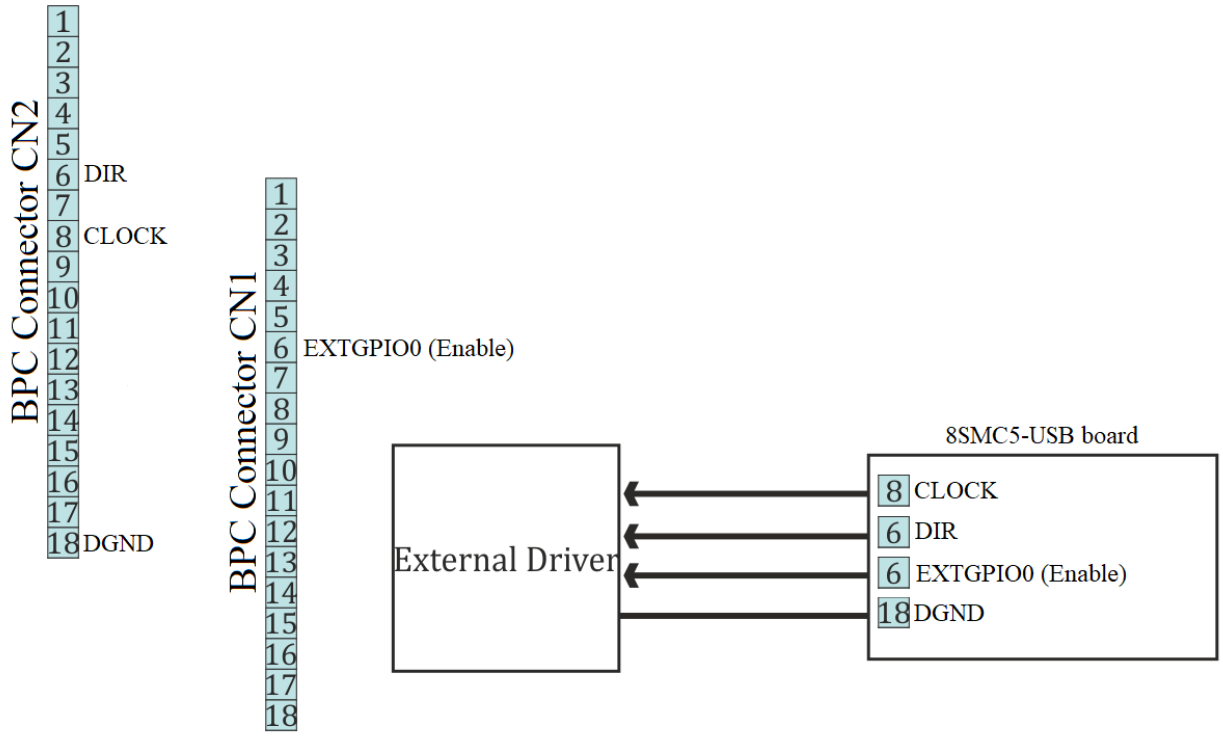
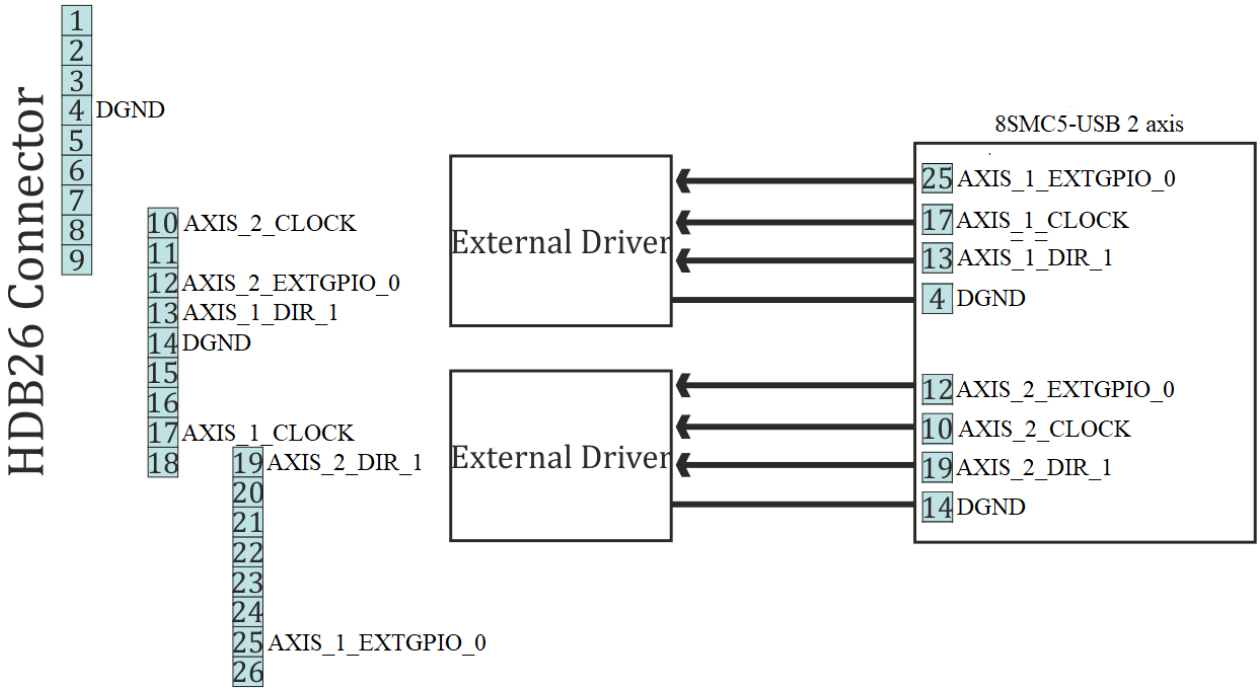


Fig. 4.50: Scheme of connection to the external driver for the controller board

4.5.9.1.2 One-axis and two-axis systems

Only *two-axis* system has the external driver interface. It outputs on the *HDB-26 connector*.

Warning: Pins 12 and 25 are general purpose inputs/outputs (see *General purpose digital input-output (EXTIO)*), but they loses functionality when external driver control is enabled.



4.5.10 Serial port

Controller allows control through UART serial port with TTL 3.3 V logic. UART outputs are located on *BPC connector*. UART outputs are located on *BPC connector*. Because of widespread availability of UART and adapters to USB, Bluetooth, Ethernet and other standard interfaces there is an option for wireless control (Bluetooth) or control over the internet (Ethernet). UART data protocol is the same as USB data protocol. I.e. it is enough to enable nonstandard serial port polling in XiLab or in libximc and the device will be found if response delay doesn't exceed two seconds. It is also possible to control controller with the help of other independently programmed micro-controller, however in this case it will be required to support *controller communication protocol*.

UART supports the following settings.

Transfer rate	9600-921600 bit/sec
Data bits	8
Parity	enabled or disabled
Parity type	even, odd, space, mark
Stop bits	1 or 2

Note: To connect to the controller via UART one should first connect to it via USB and set required speed, parity and stop bits and then save settings to non-volatile memory. Standard settings are listed in *Communication protocol specification*, use them if you cannot connect.

Table 4.13: Output and input parameters

Logic type	TTL 3.3V
Maximum data transfer speed	921 kbit
Nominal output current	5 mA

4.5.10.1 Serial port connection diagram

Warning: High data transfer speeds through long cables (**the cable must be up to 5 meters long**) in case of electromagnetic interference are impossible. If transfer errors do occur, use RC filtering circuits and reduce transfer speeds so that characteristic time of an RC-circuit is at least 4 times less than time needed to transfer a single bit. RC-circuit characteristic time is chosen based on the circumstances.

4.5.10.1.1 Controller board serial port connection

For the *controller board* UART outputs are located on the *BPC connector*.

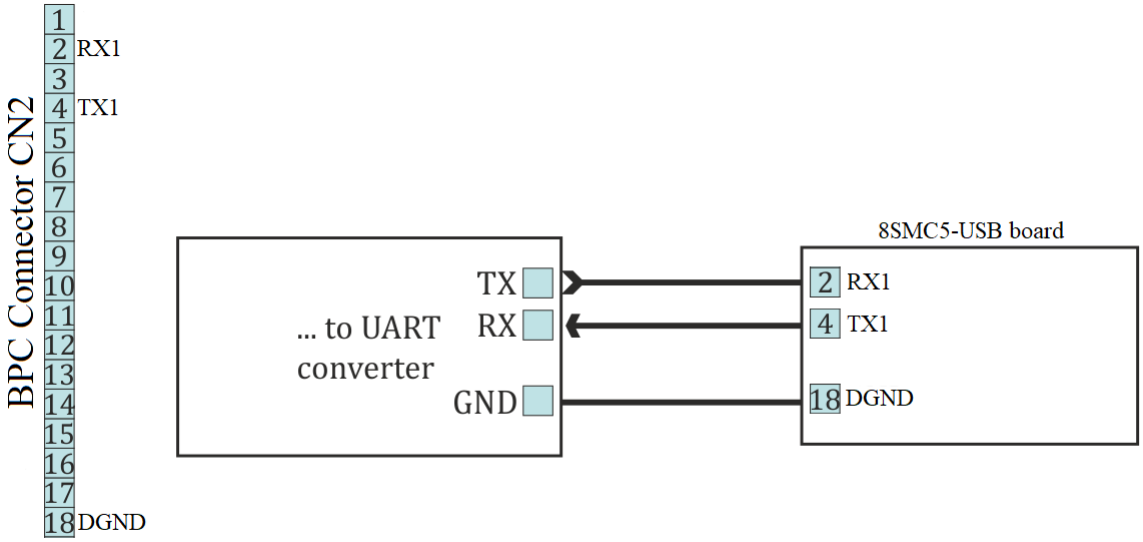


Fig. 4.51: Recommended scheme of connection to serial port pins for the controller board

4.5.10.1.2 One-axis and two-axis systems serial port connection

Two-axis and *one-axis* systems has UART outputs. Corresponding contacts for each axis output on the *HDB-26 connector*.

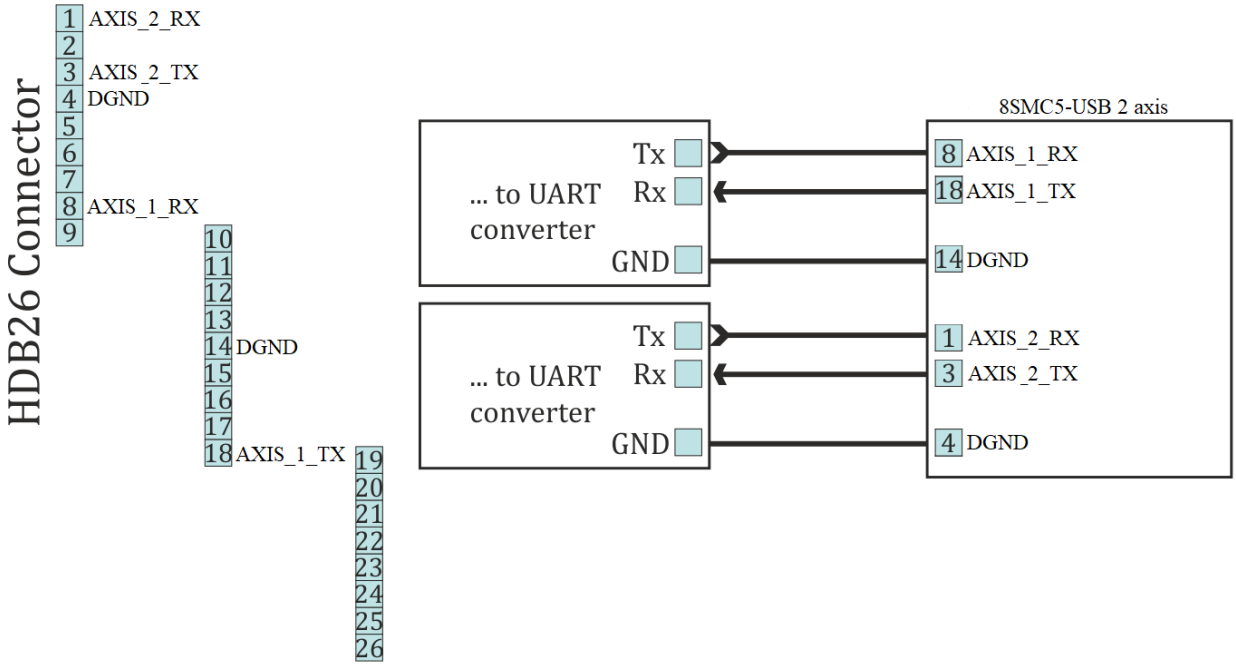


Fig. 4.52: Recommended scheme of connection to serial port pins for two-axis system

4.5.11 Saving the position in FRAM memory

Controller has a function which automatically remembers its last position. This allows one to power-off the controller after it has stopped. On the next power-on the controller will appear in the same motor position, logical position and encoder value. This will work if during the time controller was off the motor shaft was not rotated by external means.

Note: For this function to work one should wait at least 0.5 seconds after the stop before cutting the power. Position is saved even if the controller was powered off during movement, however in this case its saved position will not be exact and a new *calibration* will be needed.

4.5.12 The Standa stages detection

Newest Standa stages (please check with the manufacturer for the list of exact models) have an option to store settings and informational parameters in the internal stage memory. This chip is flashed with correct settings, which allows one to skip optimal stage configuration and to start working with the stage right out of the box. This memory chip also holds user-defined stage name (see XiLab tab *Stage name*).

When this kind of stage is connected to the controller (for more information about electrical connection please see *example of motor connection* and *stage connector*) informational parameters are automatically loaded into the controller memory, see *Stage specifications*. If the EEPROM_PRECEDENCE flag was set, which indicates precedence of settings read from external memory over the settings saved in the controller flash memory (see *About controller*), then all controller settings except UART settings and controller name are also read and applied.

If the EEPROM_PRECEDENCE flag is set then you don't need to check and/or set stage settings (for example limit switches orientation and position, nominal current, encoder and magnetic brake parameters, etc). All of this will be done automatically when a stage with internal memory chip is connected. However, if this flag is set then settings from stage memory will be loaded every time a stage with memory chip is connected and every time the controller is powered on. That is why if you need to change some settings you need to clear this flag, change required settings and save them to controller flash memory.

Note: There is a simple rule for this flag preferred state: This flag should be true on early stages of work to embrace the simplicity of automatic settings. Later, as soon as the will be need for fine tuning the settings, this flag should be set to false, not forgetting to save this to FRAM.

Note: If a stage with internal memory is disconnected from the controller no settings are changed.

4.5.12.1 For developers

Stage data is stored in DS28EC20 chip connected by 1-wire interface.

Controller periodically sends reset signal to the EEPROM chip during stage detection. If a response is received, then controller reads data from the stage into RAM, applies settings and sets STATE_EEPROM_CONNECTED bit in status structure. In XiLab this is shown by EEPR indicator in the main window. The EEPROM is then regularly polled. In case connection with EEPROM is lost (no response to the reset signal) EEPR XiLab indicator is cleared.

4.5.12.2 Connection diagram for external memory test

Outputs for connection to the memory chip are located on the *D-SUP 15 pin* connector for all systems (*controller board*, *one-axis* and *two-axis* in box.

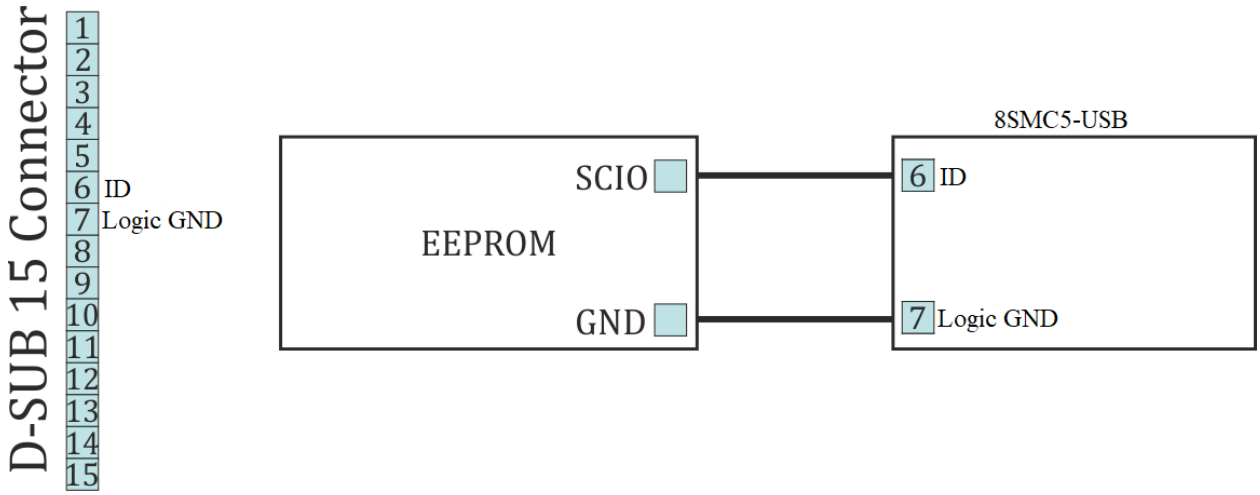


Fig. 4.53: Connection diagram for external memory test

4.6 Secondary features

4.6.1 Zero position adjustment

Controller supports setting of zero position. This function should be used for anchor marked stages, so that anchor position matches logical zero. Also, this function is convenient to use in case there is a single chosen physical position.

To set zero position a *special command* is used. This will zero step/microstep position and encoder count values. Setting of zero position happens simultaneously for all position counters and will not lead to their desynchronization. Current movement command is not affected. If controller was moving to some physical position when logical position was reset to zero by this command then the movement will still end in that physical position. For example, if the controller moved towards logical position 1000 and set zero position command was sent when it was passing 200, then logical position counter will be decremented by 200 and movement will end in logical position 800.

Note: Setting of zero position when using shift on offset (see *Predefined displacement mode*) will not change target physical position. Next shift will happen towards the same physical position which would have happened without zero position command.

4.6.2 User-defined position adjustment

A SPOS command can be used if it is necessary to set position and/or encoder value to some user-defined position instead of zero. New step/microstep position and encoder count values are passed as parameters to this command. If only one of these counters is needed one should use ignore flags to filter required fields.

This command is different from set zero command in that it doesn't set target position used by MOVE and MOVR commands to zero. During movement and stopping its behavior is the same. If you issue SPOS command during movement the controller will end in the same physical position it would move to if this command was not sent.

4.6.3 Controller status

Controller tracks its own status and can transfer it in the status structure of the *GETS* command. Controller status contains information about performed movement, its result, state of power supply, state of encoder, state of motor windings, digital input-output states, numeric information about position and powering voltage and currents and also error flags.

4.6.3.1 Movement status

MoveSts contains:

- Movement flag which is set when controller changes motor position.
- "Target speed reached" flag which is set if current speed is equal to the speed controller should be moving with.
- Backlash compensation flag, which is set during backlash compensation in the final stage of the movement (see *Backlash compensation*).

MvCmdSts contains information about the command being executed. All motor movements are initiated by movement commands to the *MOVE* target position, *MOVR* shift relative to the last target position, *RIGT* movement to the right, *LEFT* movement to the left, smooth stop *SSTP* or fast stop *STOP*, *HOME* home position calibration and *LOFT* forced backlash compensation. Control by buttons, joystick, sync in pulses, etc. is also performed by these commands. For example, joystick calls right and left movement commands during deflection or smooth stop command in central position (see *Joystick control*). Current movement command or last command and command status (running/completed) are located in MvCmdSts variable. If the command is completed then another bit shows its result (successful or not). Unsuccessfully completed command means controller could not reach desired position or backlash compensation could not be performed. The reason for this can be a sudden stop due to limit switches or Alarm state. Initial state of this field contains unknown command and successful completion status.

4.6.3.2 Motor power supply status

PWRSts contains information about supply voltage. Windings' status can be:

- Disabled (in this case no voltage is applied).
- Powered by reduced current relative to nominal current (for example if winding current reduction option is used).
- Powered by nominal current.
- Powered by an voltage insufficient to reach nominal current in the windings.

Last status frequently appears with high rotation speeds, because for higher step switching speed one needs higher voltage to ensure current rise in motor winding inductance. Insufficient voltage does not mean the motor won't move, it will merely emit excess noise and its torque will drop (see *Power control*).

4.6.3.3 Encoder status

EncSts contains information about connected encoder if feedback is disabled (for example for stepper motors). Encoder state can be one of the following:

- Not connected
- Unknown state, when there is not enough data to define encoder state.
- Connected and working.
- Connected and reversed, in this case it is necessary to enable reverse in encoder settings.
- Connected and defective.

The last state is realized when switch signals come to encoder inputs but they don't correspond to the motor rotor movement. State change happens after sufficient statistical data is collected. That's why detection doesn't happen immediately. It is also impossible to define encoder status without movement (see *Operation with encoders*).

4.6.3.4 Motor windings status

WindSts contains information about windings state. State of each of the two windings is shown separately. They can be:

- Disconnected from controller
- Connected
- Short-circuited
- In an unknown state.

A state with very small resistance and inductance is considered to be a short-circuit. A state with very high load resistance is considered to be disconnected.

4.6.3.5 Position status.

All data about stage position and speed is reflected in status structure. Fields of primary position (CurPosition, uStep), secondary position (EncPosition), speeds (CurSpeed, uCurSpeed) are used for this. Primary position is counted in steps and microsteps of stepper motor if control without feedback is used. In case of leading encoder mode

encoder counts are stored in CurPosition and uStep contains 0. Secondary position contains encoder coordinate if no feedback is used for stepper motor, contains steps if a stepper motor with encoder feedback is used and contains 0 if DC motor is used. Speed is always displayed for the primary position and is measured in the same units as the current set speed.

4.6.3.6 Controller power supply status and temperature.

Status structure reflects:

- Power current (in mA)
- Power voltage (in tens of mV)
- USB current (in mA)
- USB voltage (in tens of mV)
- Microprocessor temperature (tenths of degrees Celsius)

4.6.3.7 Status flags

There are several types of flags: control command error flags, critical parameter flags, general error flags and state flags.

Note: Many flags do not remove themselves and should be reset by the *STOP* command.

Protocol command errors:

- errc – Unknown protocol command. This error should not appear if the used software corresponds to the used controller protocol version. Flag can't be removed by itself.
- errd – Data integrity command check code is incorrect. This error appears in case of data transfer failure. The flag can't be removed by itself.
- errv – One or more values sent in the command could not be applied. It appears when command was received and successfully recognized but transferred data were incorrect or out of range. This error can also mean that necessary operation is impossible because of hardware failure. For example, this error appears if you set microstep mode which is not in supported list or if you set zero steps per motor revolution. The flag can't be removed by itself. Critical parameter exceeded errors:
- Flag which means that controller is in Alarm state.
- Flag which means that power driver gives overheat signal. The flag is removed by itself depending on *critical parameters settings*.
- Flag which means that microprocessor temperature is out of acceptable range. The flag is removed by itself depending on *critical parameters settings*.
- Flag which means that power supply exceeded acceptable value. The flag is removed by itself depending on *critical parameters settings*.
- Flag which means that power supply voltage is lower than acceptable value. The flag is removed by itself depending on *critical parameters settings*.
- Flag which means that current drawn from the power unit exceeded acceptable value. The flag is removed by itself depending on *critical parameters settings*.
- Flag which means that USB voltage exceeded acceptable value. The flag is removed by itlsef depending on *critical parameters settings*.
- Flag which means that USB voltage is under acceptable value. The flag is removed by itself depending on *critical parameters settings*.
- Flag which means that current drawn from the USB exceeded acceptable value. The flag is removed by itself depending on *critical parameters settings*.
- Flag which means that limit switches are mixed up. The flag can't be removed by itself.

General error flag:

- Flag which means that position control system detected steps counter and position sensor desynchronization. The flag can't be removed by itself (except the case of using position correction).

State flags:

- Presence of connected stage equipped by EEPROM memory.
- Presence of external power supply. Otherwise power supply is internal. Is always set.

4.6.3.8 Digital signals status.

Controller reflects input and output digital signal status as active state flags or as current logical level. Active state corresponds to one or to zero depending on specific block settings, for example on inverting settings. Flags can be:

- Right limit switch state (one if limit switch is active).
- Left limit switch state (one if limit switch is active).
- Right button state (one if button is pressed).
- Left button state (one if button is pressed).
- 1 if EXTIO pin operates as output. Otherwise - as input.
- EXTIO pin state (1 if state is active on input or on output).
- Hall A sensor state (1 if logical one is on input).
- Hall B sensor state (1 if logical one is on input).
- Hall C sensor state (1 if logical one is on input).
- Magnetic brake state (1 if power supply is applied to brake).
- Complete revolution sensor state (1 if sensor is active).
- Input synchronization pin state (1 if synchronization pin is in active state).
- Output synchronization pin state (1 if synchronization pin is in active state).
- Input encoder A channel state (1 if logical one is on input).
- Input encoder B channel state (1 if logical one is on input).

4.6.4 USB connection autorecovery

This unit is designed to reboot the USB in the event of loss of communication (for example, this may occur in the event of electrostatic discharge or when the USB is disconnected without powering down the controller). The on/off state of this unit is determined by the *USB_BREAK_RECONNECT* flag (see *Critical parameters*). If the unit is turned on, it monitors the connection loss on the USB. In the case of communication loss on the USB after 500 ms the firmware reconnects the device and then checks the state of the USB bus. If for a certain time there is no recovery of connection (i.e. data communication), then this unti reconnects the USB again. Thus, in case USB connection is not restored, the controller will continuously reconnect to the USB bus until connection is restored or until the time between reconnection attempts exceeds 1 minute. So, in the case the USB is disconnected without powering down the controller (for example, in the case of motor control with buttons or joystick) controller will remain in USB reconnection mode for about 5 minutes.

Note: USB reconnection mode does not affect other controller functions (for example movement or winding current maintenance) in any way.

To avoid simultaneous reconnect to the USB bus from both the controller and the computer side, the time between the reconnections changes exponentially (see *Time between USB reconnections*).

Table 4.14: Time between USB reconnections

Restart number	timeout, ms
0 (after communication is lost)	500
1	483
2	622
3	802
4	1034
5	1333
6	1718

The status of the unit can be determined by LED flashing frequency. In the case controller is in reconnection mode the LED will flash with a frequency of 10 Hz (see *Operating modes indication*).

Warning: Because of the structure of the program unit, as well as USB bus specification, unit doesn't guarantee 100% recovery of the communication with the computer after a static discharge.

Xilab software also tries to reconnect to the controller when it is running. On connection loss, which is defined as "result_nodevice" libximc library call error, Xilab waits for 1000 milliseconds, then attempts to reopen device port. On Windows operating systems Xilab uses WINAPI functions to check if corresponding COM-port device is present. If it is, then after two unsuccessful attempts to reopen it calls libximc ximc_fix_usbser_sys function, which resets the usbser.sys driver to fix the driver error. On Linux or MacOS Xilab simply tries to reopen the device every 1000 ms. After the device is opened Xilab sends several commands to read serial number, firmware version and controller settings which are needed to set up user interface.

Libximc library considers device lost (return error code result_nodevice) on critical errors from system calls ReadFile/WriteFile (Windows OS) or read/write (Linux/Mac OS).

4.7 Software compatibility

4.7.1 MicroManager

4.7.1.1 Preparation

Important: This guide for MicroManager was developed and tested on version 2.0-gamma. The guide for earlier versions can be found on the Internet.

- Download and install MicroManager. This is a simple procedure, and there are usually no problems with the setup process.
- Download the libximc-2.7.6 library distribution

Warning: MicroManager will **only work** with the libximc library version-2.7.6. It will **not work** with any other libximc version!

- Copy following DLLs from extracted directory *ximc-2.7.6/ximc/winX/* to the MicroManager directory: *libximc.dll*, *xiwrapper.dll*, *bindy.dll*. **The bitrate of the copied libraries must match the bitrate of the installed MicroManager.**
- Install VC++ 2013 Redistributable Package from the extracted file *ximc-2.7.6/ximc/winX/vcredist.exe* **The bitrate of "vcredist" must match the bitrate of your OS.**

- Connect a power supply to *controllers* and set the appropriate voltage for motors in your stage. Turn on a power supply.
- You need XiLab to check your *controllers* controllers and set it. XiLab version depends on the controller firmware version. You can download XiLab and update controller's firmware at this page:
- Connect controllers to computer via USB and open them in XiLab software. Click *Load setting from profile...* in *Settings...* window of XiLab and choose appropriate profile for your stage. Click *Save settings to flash*. For additional information visit: *XILab application User's guide*.

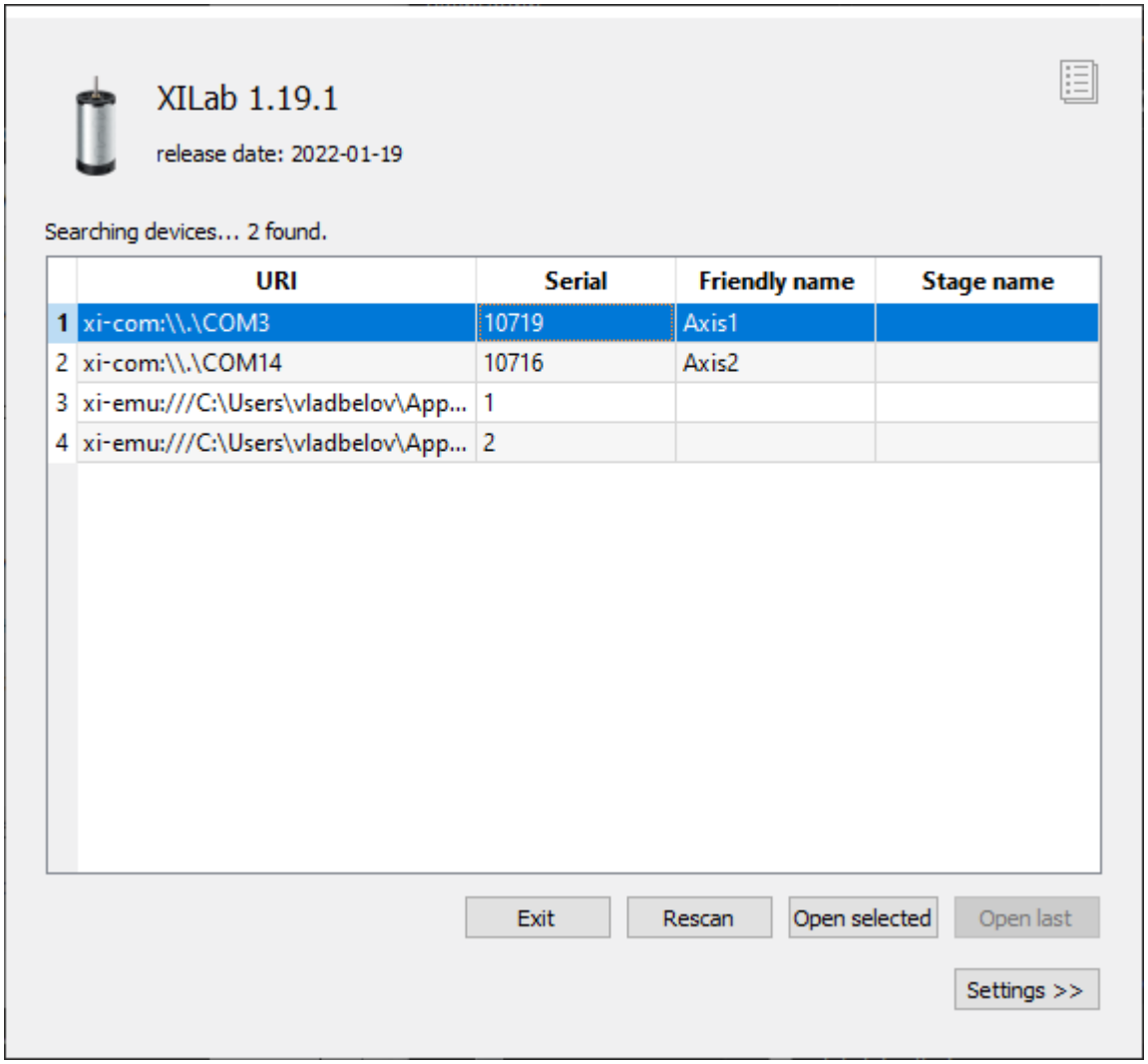


Fig. 4.54: Example of controller discovery in xilab

4.7.1.2 Getting started with MicroManager

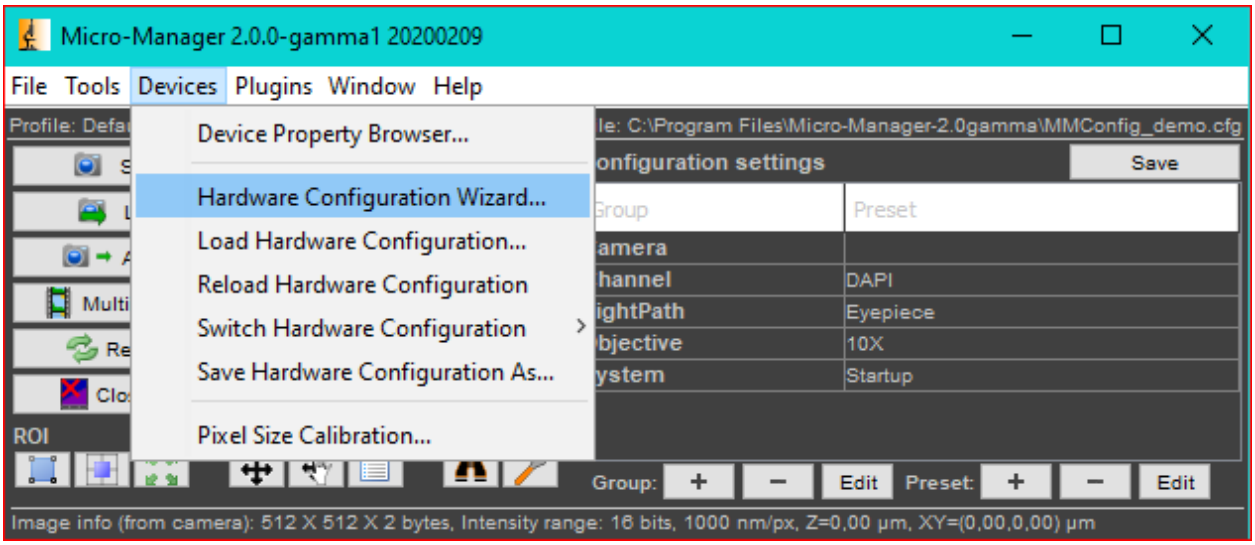
4.7.1.2.1 Run MicroManager

Note: When installing the MicroManager in **C:\Program Files** access to configuration files is limited and you need to have administrator privileges. To start an application as administrator right click by its icon and choose *Run as administrator*.

- Run MicroManager from the shortcut at your desktop or start *ImageJ.exe* application from installation directory. First time it will greet you and suggest to type some information about yourself.
- The next window contains drop-down list with configuration files. Choose *None*.

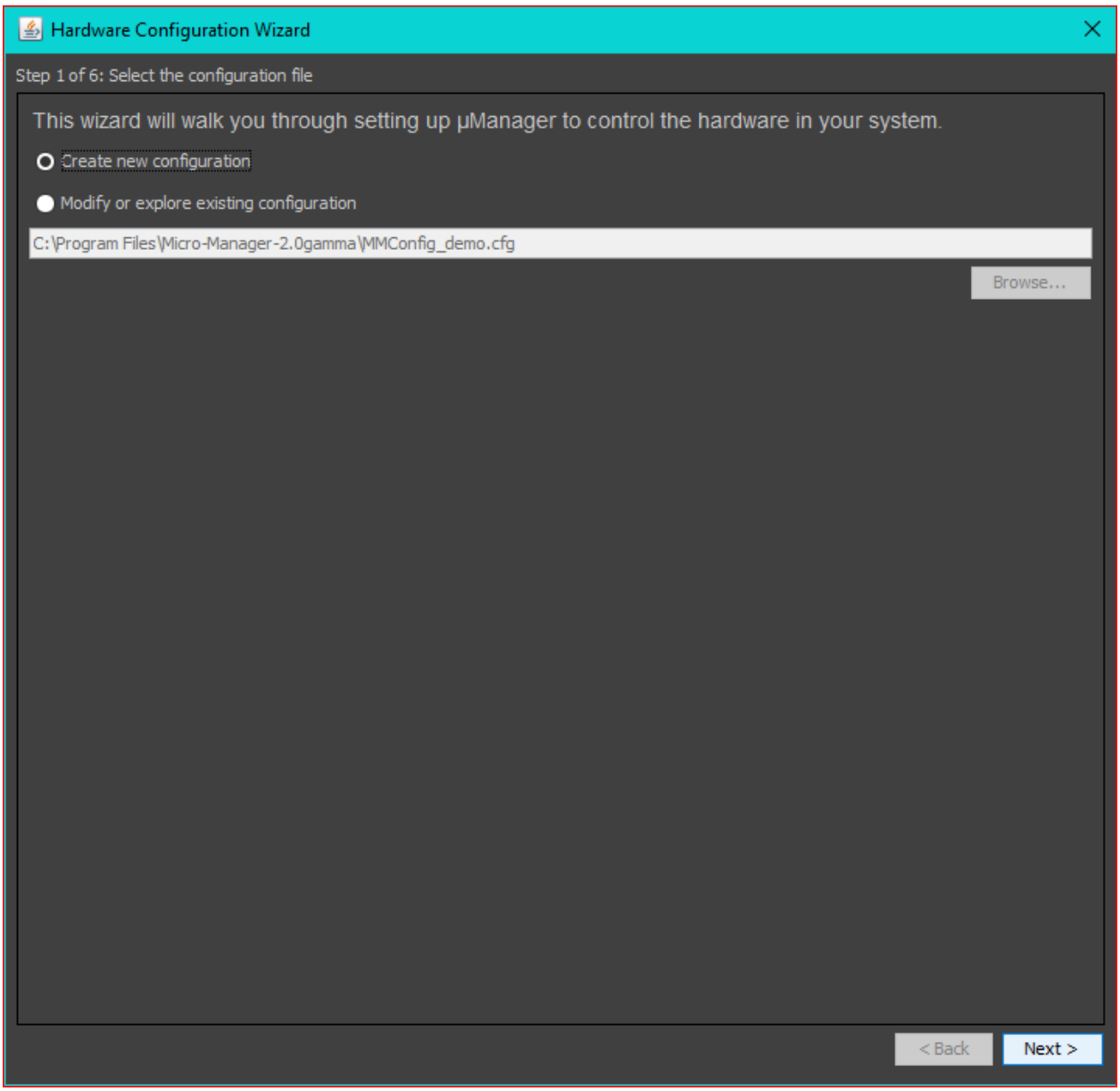
4.7.1.2.2 Configure hardware

- In the main window choose *Devices* → *Hardware configuration wizard*.

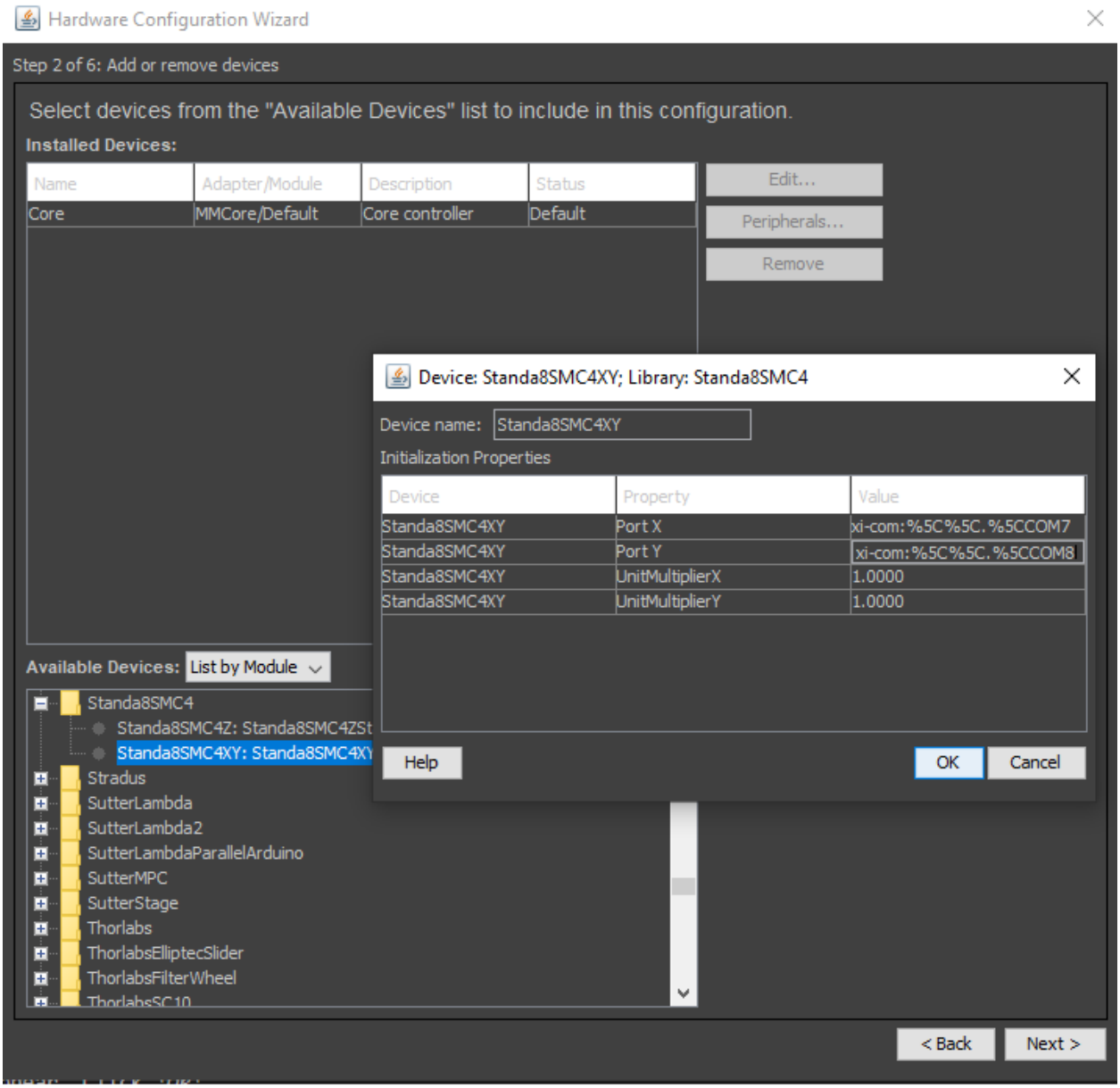


The process of hardware configuration consists of 6 steps:

1. At start, choose “Create new configuration” and click “Next”.



2. At the second step MicroManager prompts you to add hardware. If you want to use MicroManager with *8SMC4/5-USB* motor controller choose *Standa8SMC4* folder in down window with available devices. There are two drivers: *Standa8SMC4Z* for motor controller with one axis and *Standa8SMC4XY* for controller with 2 axes. When you select one of them, a dialog box with properties will appear.



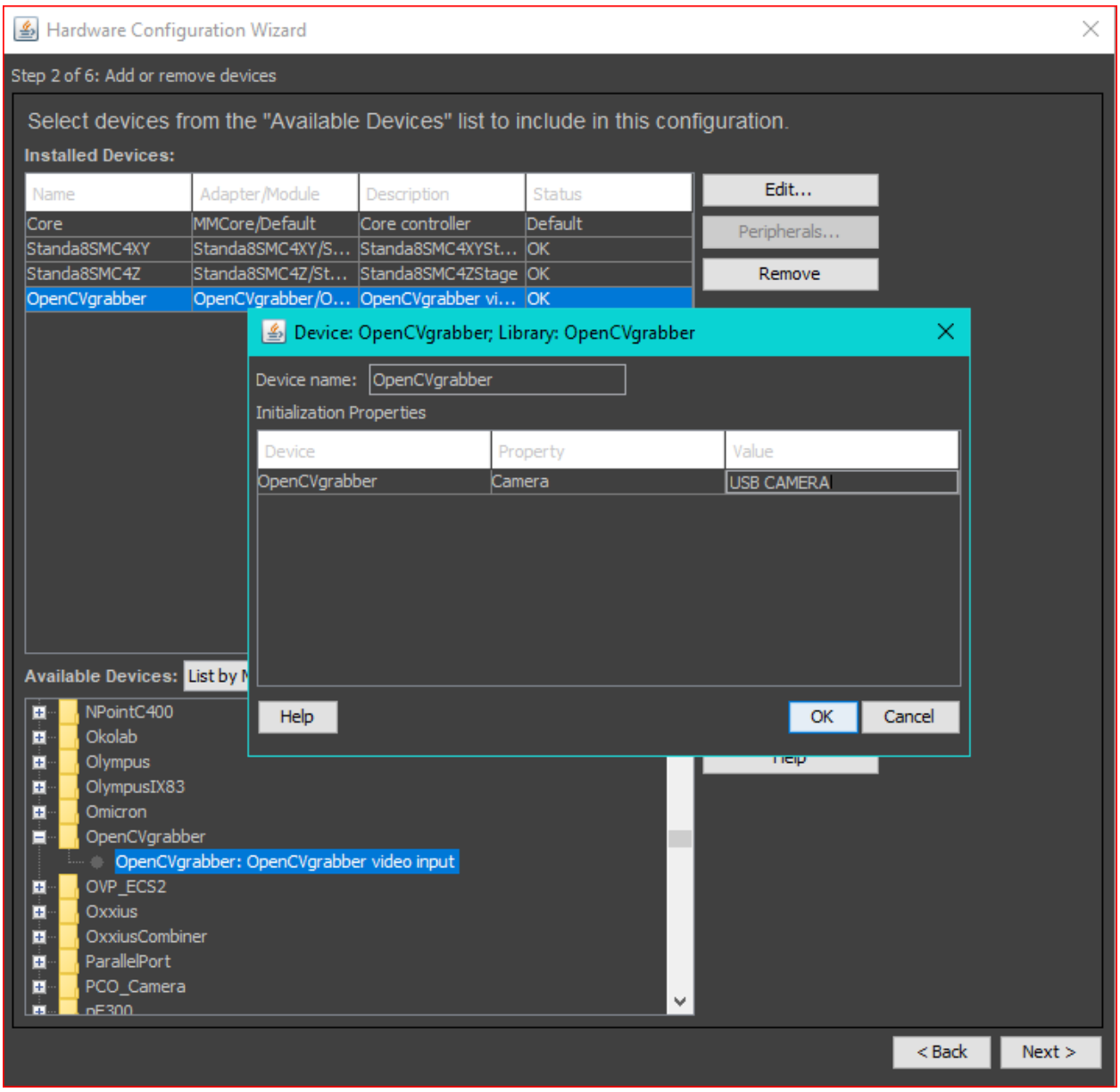
3. In the *Value* column type the COM port number for motor controller in the next format:

```
xi-com:%5C%5C.%5CCOMn
```

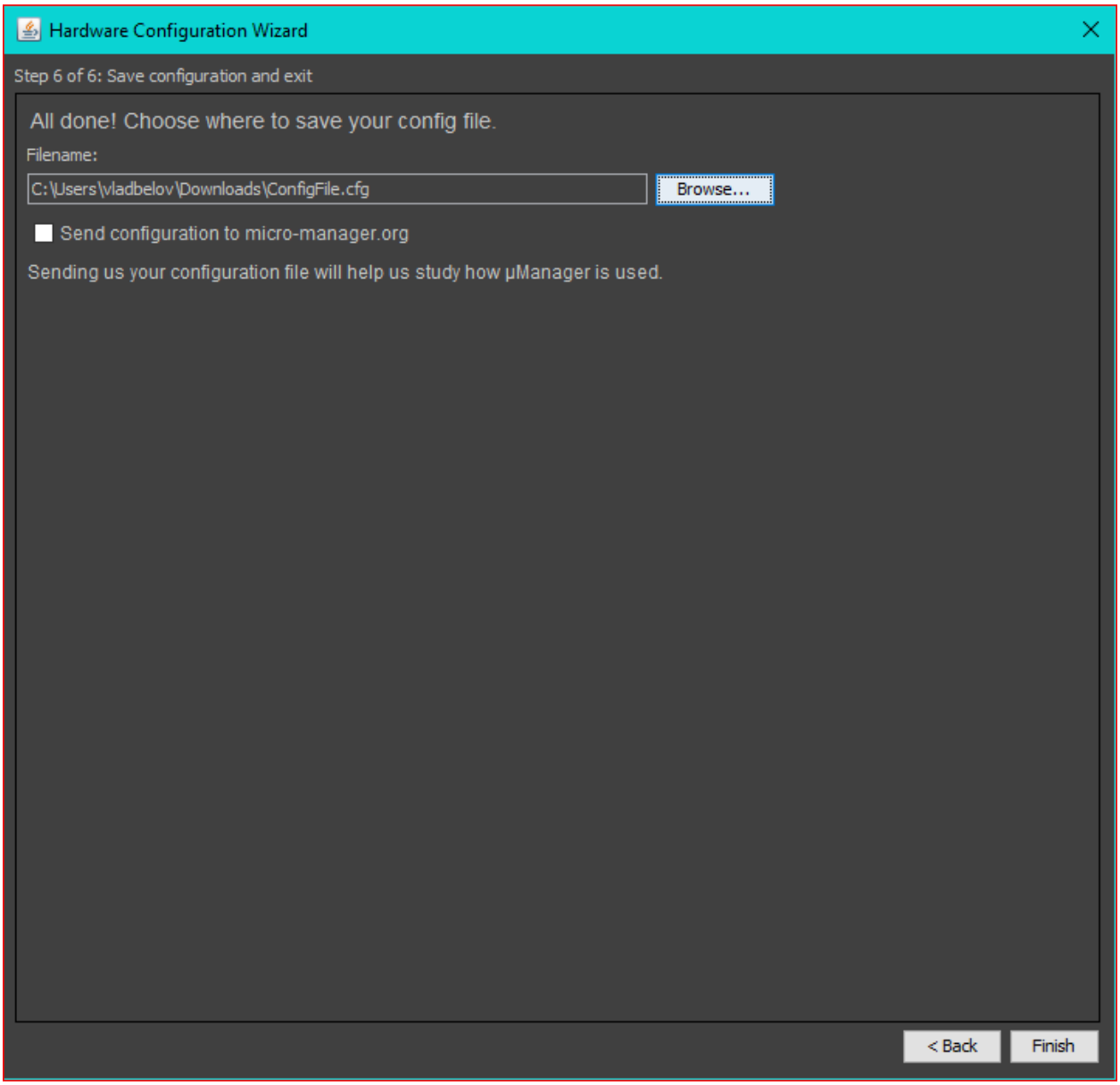
   Trailing letter *n* is a number of corresponding COM port (look at the screenshot above). You can find it in XILab or in *Device Manager* → *Ports*. Click *Ok*. *Unit Multiplier X/Y* fields are leaved with default values. These values allow you to set stage speed.

4. For *Standa8SMC4Z* you should type COM port and Unit Multiplier too. Z axis is usually used as a focusing stage.

5. Camera is added in the same way. You need to know correct driver name for your camera. *OpenCVGrabber* driver may be used for any USB camera. Choose OpenCVGrabber folder in Available Devices. The next window will appear. Click *Ok*.



6. Click *Next* continue, choose where to save your config file. After click *Finish* in the last step.



The system configuration is finished.

4.7.1.2.3 Device usage

We have installed and configured MicroManager and can check how does it work.

1. Set the pixel size parameter in *Devices* → *Pixel Size Calibration...*. There are you should type the pixel size value and mark parameters that MicroManager should consider. Click *New* to create new pixel size. In appeared window mark resolution of your camera (with *OpenCVGrabber* it is *OpenCVGrabber - Resolution* parameter) and type pixel size (in um). If you want to calculate pixel size parameter you need to know image size from microscope, its resolution and displacement (from screw pitch) for you stage.
2. Choose *Devices -> Stage Position List...*. Click *Set Origin* to set zero of your stage coordinate system. Then, in the main window click *Live*. Choose *Scrolling Tool* (like a hand) in the main window. Make sure that cursor looks like a hand on the screen. Click the windows with the video from camera. Now you can control your stage by keyboard (arrow keys to control XY axis and U,J buttons to control Z axis).

3. You can scroll an image from a camera with the mouse cursor. Click *Tools* → *Mouse Moves Stage* (Use Hand Tool). Now click video window and drag it in the desired direction. Stage will move in the same direction. Double click will allow you to center the view on the chosen point in the image.

Note: For example you set the pixel size is equal to 1um. And *UnitMultiplierX/Y* is equals to 1.0000. With 8MTF stage table one pixel shift corresponds to 12.5um. If we will set *UnitMultiplierX/Y* to 12.5 than movement at one pixel will corresponds to 1um.

Note: You may decrease *UnitMultiplier* if the stage speed is low and increase it if the stage speed is very high.

4.7.2 TANGO

Note: The **8SMC4-USB-Eth** adapter has been renamed **8Eth1**[1]. All available instructions in our manual are relevant and applicable to both devices.

4.7.2.1 Overview



Fig. 4.55: Our controllers provide TANGO interface!

The TANGO control system is a free open source device-oriented controls toolkit for managing any kind of hardware or software and building SCADA systems. It is used for controlling synchrotrons, lasers, physics experiments in over 20 sites. It is being actively developed by a consortium of research institutes.

TANGO is based on the concept of Devices. Devices implement object oriented and service oriented approaches to software architecture. The Device model in TANGO implements commands/methods, attributes/data fields and properties for configuring Devices. In TANGO all control objects are Devices and hardware access is managed in a process called a Device Server.

The Device Server contains Devices belonging to different Device Classes which implement the hardware access. At Device Server startup time Devices (instances of Device Classes) are created which then represent logical instances of hardware in the control system. Clients import the Devices via a TANGO Database Server and send requests to the devices using zeromq and CORBA protocols.

Standa 8SMC5-USB controllers are compatible with TANGO 8.1 system via custom device server implementation and can be used in two scenarios:

[1] The vendor code was changed in 2020, earlier this product had the vendor code 8SMC4-USB-Eth.

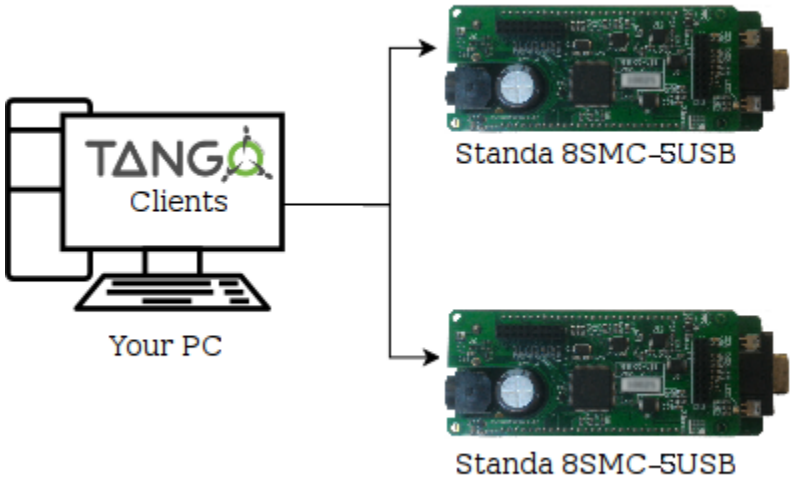


Fig. 4.56: Using TANGO interface via device server directly on your PC

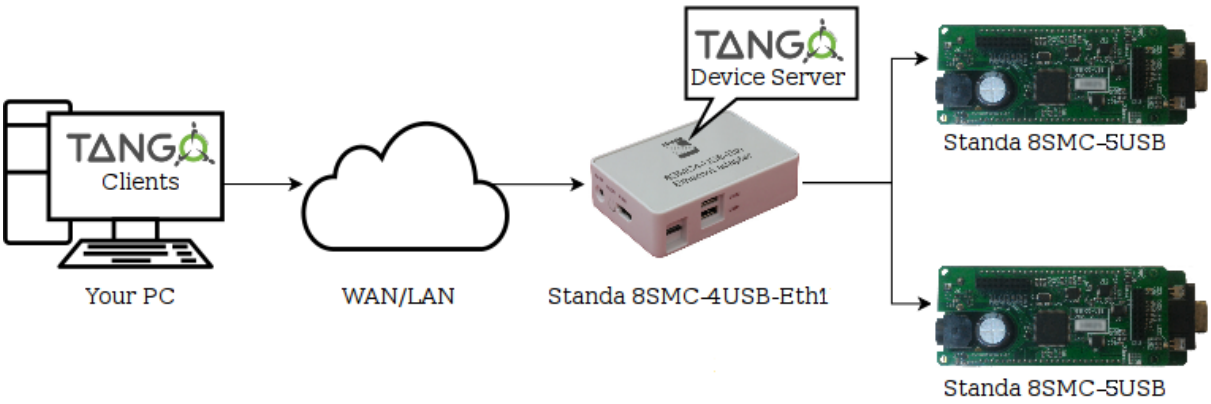


Fig. 4.57: Using TANGO interface via *Standa 8Eth1* adapter device

4.7.2.2 Declaring the device, configuring and starting device server

Warning: The device server for **Standa 8SMC5-USB** was developed and tested with TANGO 8.1 version. It *should* work with any TANGO version greater than or equal to 7 due to backward and forward compatibility in the API but hasn't been verified yet.

First of all you should declare the device in your local TANGO Database Server. The easiest way to do it is via Jive:

Important: The video on setting up a new TANGO-compatible device **shows the same steps as in the instructions**

below

https://youtu.be/PshuDGwv-LY

Video guide to setting up a TANGO

1. Connect to your local TANGO control system (*Edit -> Change Tango Host*).
2. Start registration wizard (*Tools -> Server Wizard*).



Fig. 4.58: Jive server wizard dialog

3. Enter “Standa8smc4usb” in *Server Name* field. *Instance name* can be arbitrary word containing alphanumeric symbols only (e.g. “TestDevice”).
4. Press *next* - you should see *Start the server* window. Now it’s time to actually start the device server.
    - If you have *Standa 8Eth1* and want to use its inbuilt TANGO support you should open *administration interface* and do configuration and/or perform startup from there.

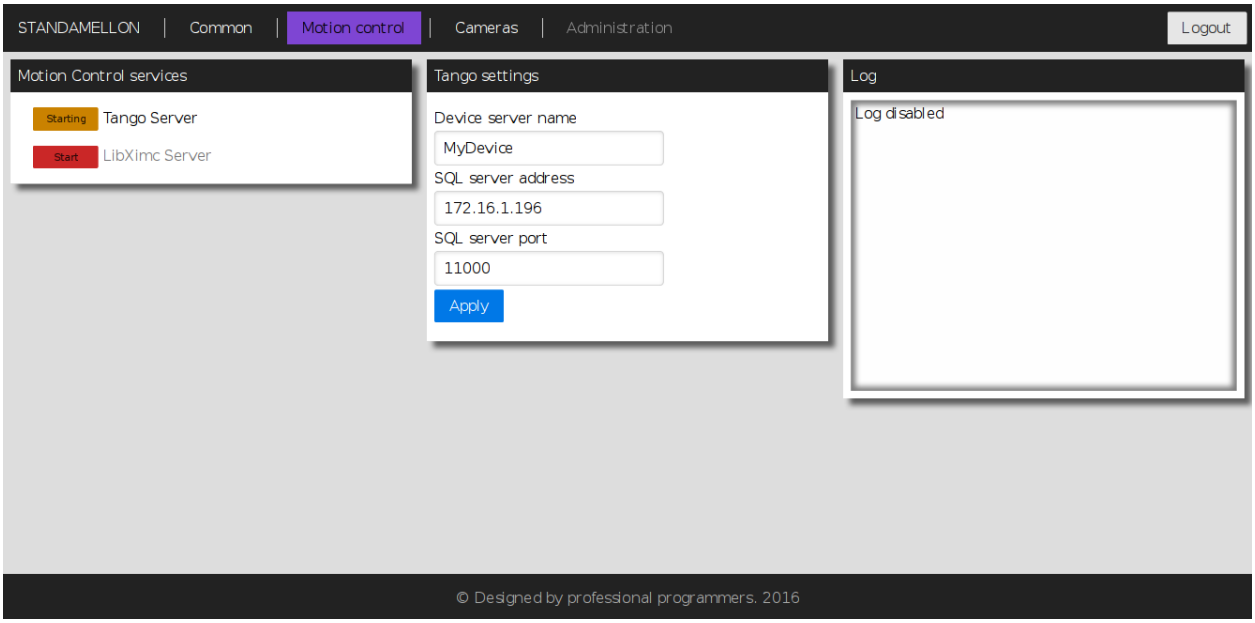


Fig. 4.59: TANGO service configuration page in the *administration interface*

- Otherwise if you're going to use TANGO device server directly, you have to compile it for your target platform manually from the source. The resulting *Standa8smc4usb* executable should be started with previously entered in "Jive" instance name as its' first argument (e.g. "TestDevice"). If everything is fine you'll see *Ready to accept requests* in standard output stream.

Note: Device server instance name entered in Jive must match the name specified during device server instance startup (both manually via executable and from *administration interface*).

Note: Before starting "Standa8smc4usb" device server make sure that "TANGO_HOST" environment variable points to your local TANGO Database Server e.g. "192.168.0.172:11000".

---

5. Go back to Jive *Start the server* window and press *next* button.
6. In the *Class Selection* window choose *Standa8scm4usb* class and press *Declare device* button.
7. Enter TANGO specification compliant *Device name* (e.g. "a/b/c") and press *next*.
8. The next step requires entering *SerialNumber*. It can be obtained either from *Common* section in the *administration interface* if you're using *Standa 8Eth1* device, or via *XiLab* (*Settings -> About Device -> Serial number*).

Note: XiLab may also be used to obtain serial number of the controller connected to *Standa 8Eth1* device but LibXimc motion control service has to be enabled in administration panel for this to work. For more information please refer to *XILab documentation*.

---

9. After that *CalibrationRatio* and *CalibrationUnits* parameters should be set. Theirs purpose is to replace internal controller coordinates with units familiar to the user and should be treated as "1 step = *CalibrationRatio CalibrationUnits*" formula. If you don't need calibration, set these parameters to "1.0" and empty string respectively.
10. Device declaration procedure (previous 3 steps) should be repeated for each connected controller you want to access via TANGO interface.

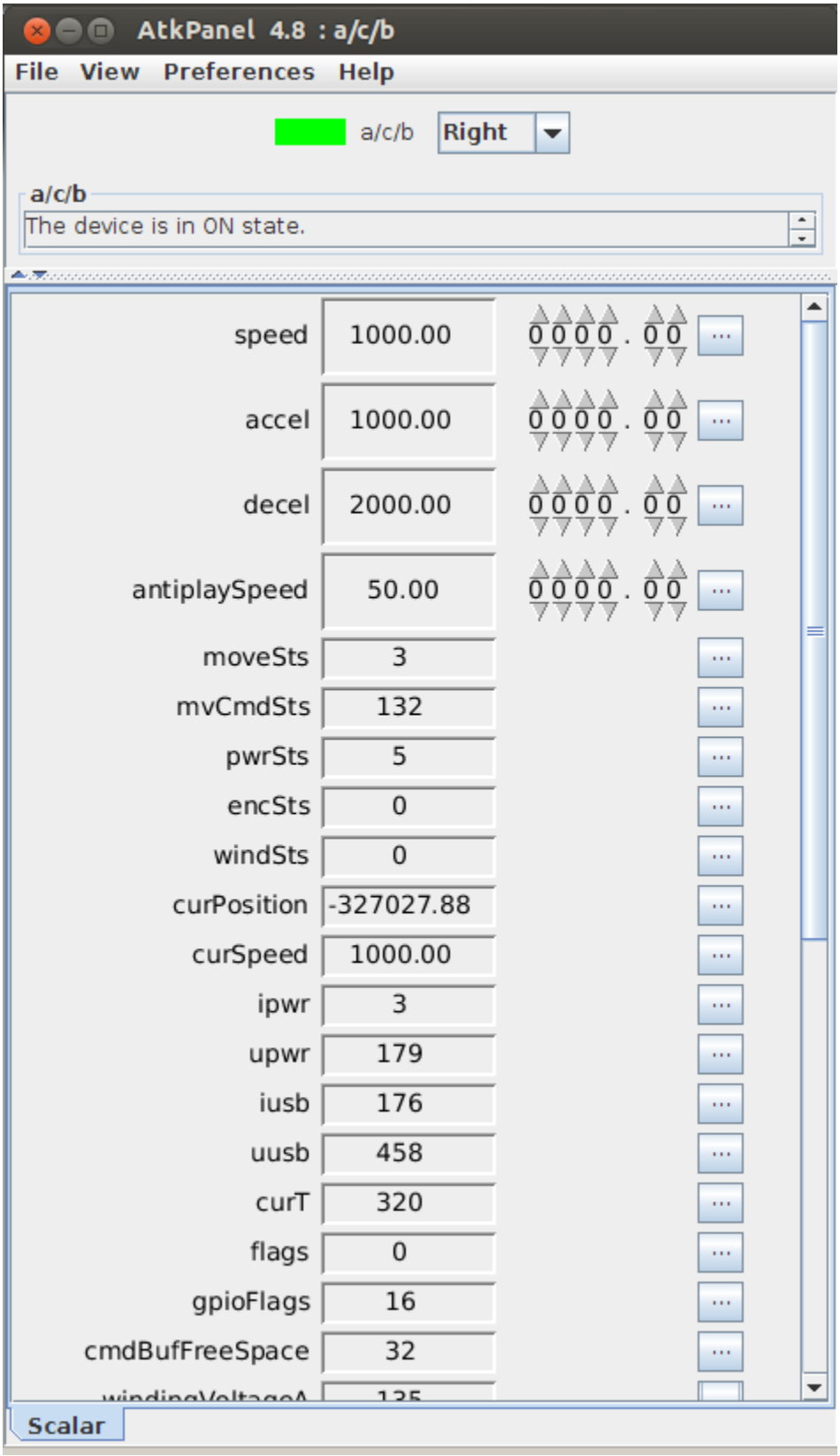


Fig. 4.60: Default ATK panel for controller exposed via TANGO

After that any exported controller is available for configuration and control through TANGO interface making it blend seamlessly with your existing infrastructure! For more info please refer to official TANGO documentation.

CHAPTER

FIVE

XILAB APPLICATION USER'S GUIDE

5.1 About XILab

XILab features a user-friendly graphical interface, which is designed for stages control, diagnostic and fine tuning of the motors driven by the controllers. XILab allows quick adjustment of connected stage by loading of previously prepared configuration files. The control process can be automated with script language that can be used either directly or to speed up the process of customized control program development. XiLab supports multiaxial mode and multidimensional control scripts. It is possible to output the data about controller and motor status in form of charts and save them to a file, or export tabular data for external processing. The software is compatible with Windows XP SP3, Windows Vista, Windows 7, Windows 8, Windows 10, Windows 11, Linux, MacOS for intel and Apple Silicone (via Rosetta 2). Depending on the OS of your computer, appearance of some windows may vary.

Here you can find the Quick Installation Guide for the application. This chapter provides a detailed manual for the XILab software.

5.2 Main windows of the XILab application

5.2.1 XILab Start window

When started, XILab opens a controllers detection window. By means of libximc library, XILab queries controllers connected to the system and displays a list of found and successfully identified controllers.

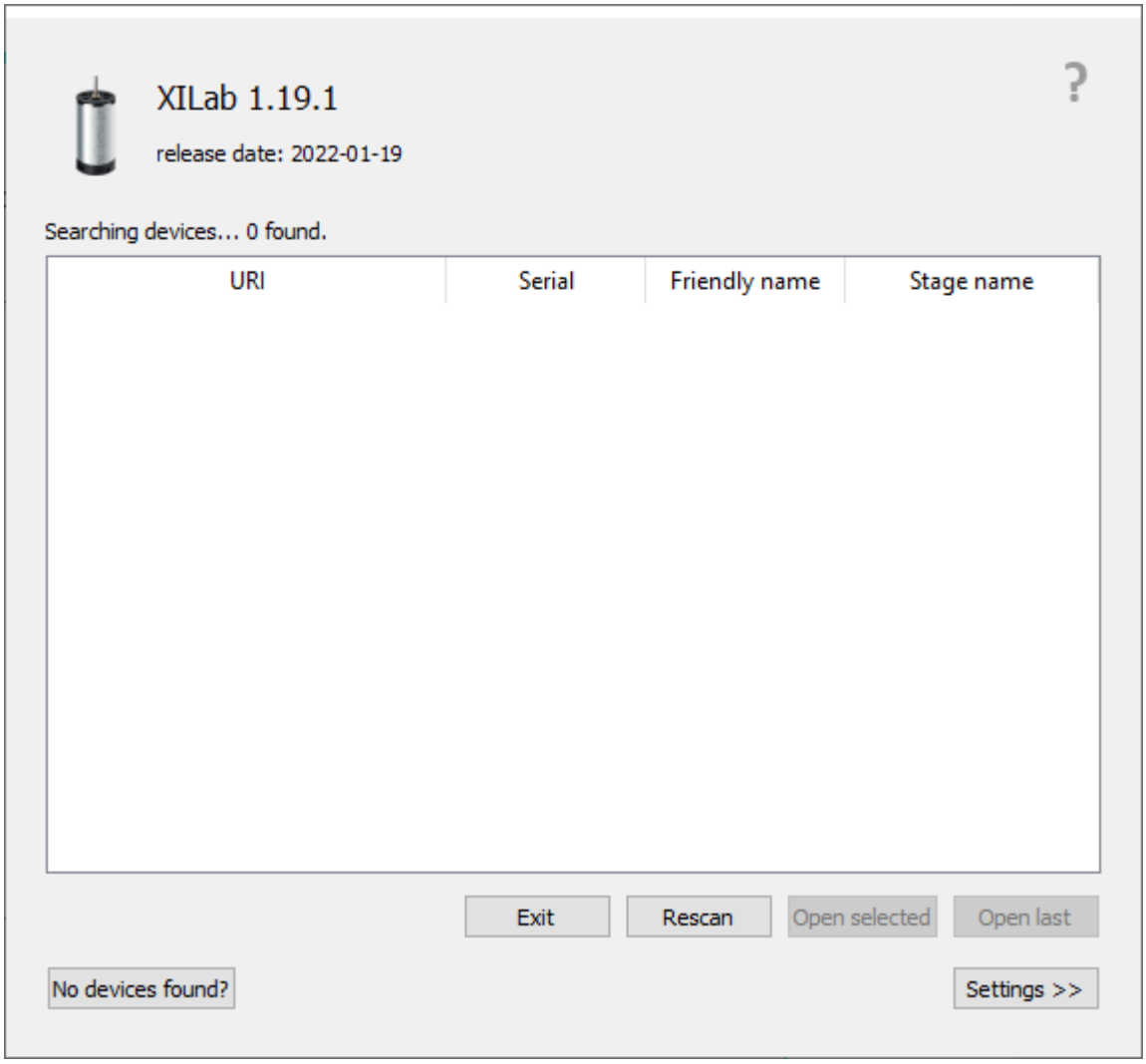


Fig. 5.1: XILab Start Window, 0 controllers found

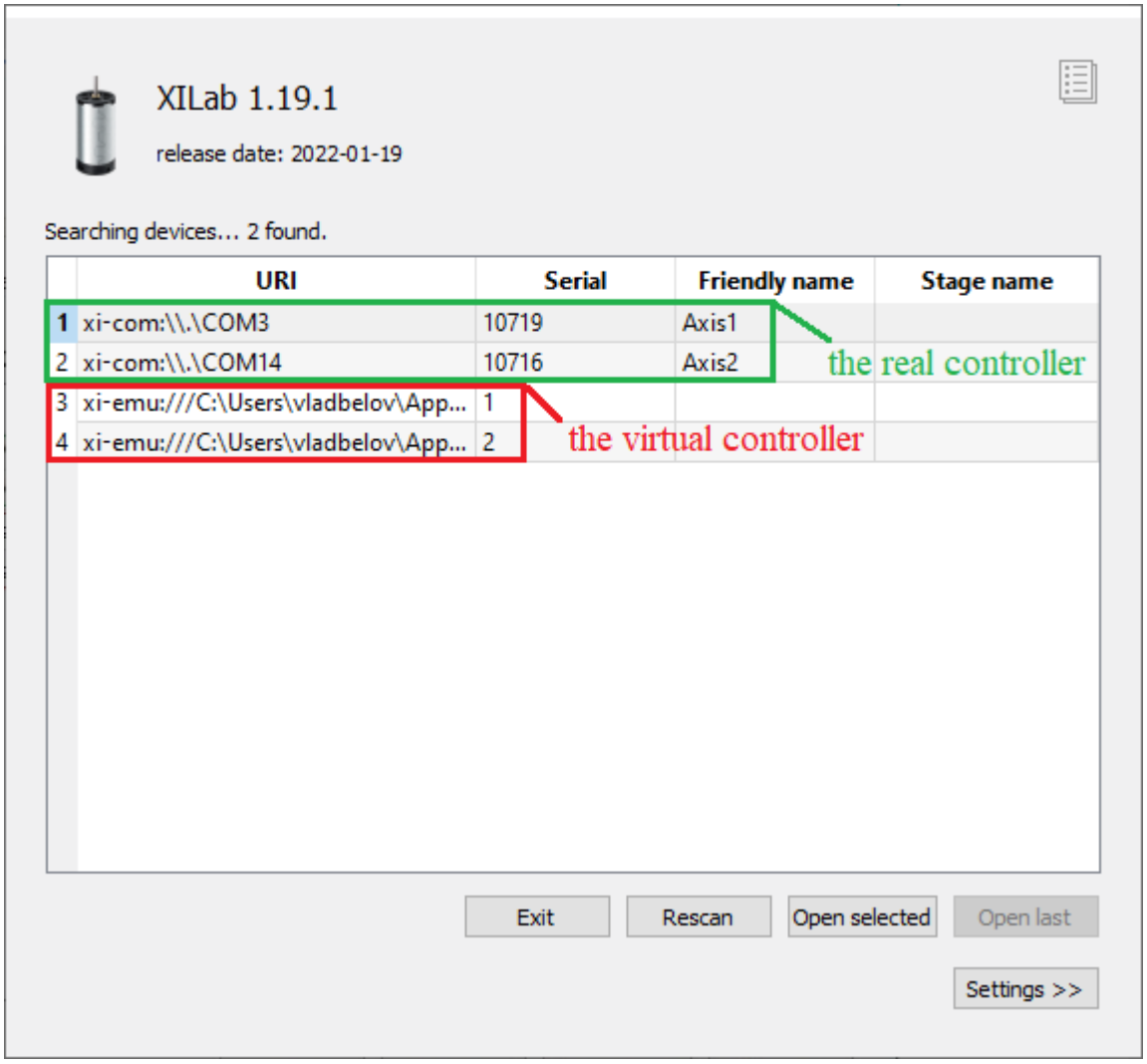


Fig. 5.2: XILab Start Window, 2 real controllers and 2 virtual controllers are displayed

The list of found controllers is displayed on the start screen. Here you can select one or more controllers and open them using the *Open selected* button. If one controller is chosen, then *XILab Main window in single-axis control mode* will be opened, if more than one controller is chosen the *XILab Main window in multi-axis control mode* window will be opened. You could repeat the search by clicking the *Rescan* button or exit the program by clicking *Exit*. If the *Open last* button is active it means that all the controllers that had been opened in the previous run of XILab were found. Clicking the *Open last* button will then open the last saved configuration.

XILab can work with virtual XIMC controllers, which support the request-response protocol of a real controller. Virtual controllers may be useful for testing and getting used to the XILab interface, if no real hardware controllers are connected to the system.

The *Settings* button opens the new tab that contains the XIMC devices detection options.

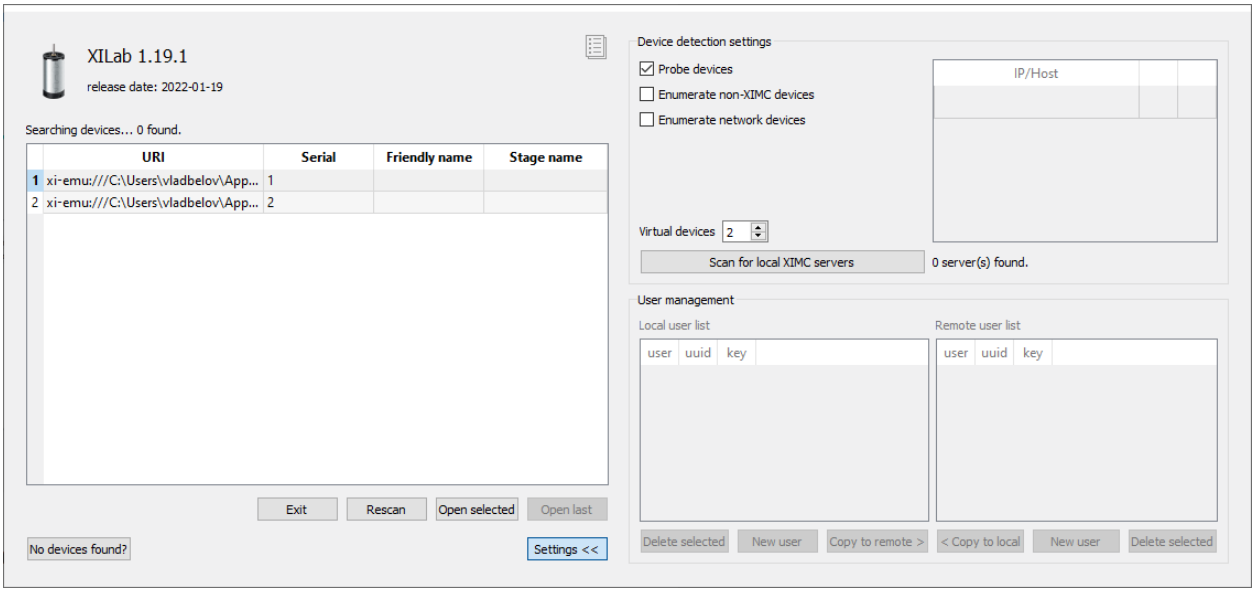


Fig. 5.3: XILab Start Window, the settings tab

Device detection settings group includes XIMC device detection settings.

If *Probe devices* option is checked, at the start application tries to identify controllers by sending them commands GETI and GSER.

If *Enumerate non-XIMC devices* option is checked the application queries all COM-port type devices in the system. If the option is disabled, only devices with names matching the XIMC mask ("XIMC Motor Controller" in Windows; /dev/ximc/* and /dev/ttyACM* in Linux/Mac) are queried.

If *Enumerate network devices* option is checked the application queries network-attached devices. A list of domain names and/or IP addesses with XIMC server software is located below. One can add entries to the list manually or use automatic detection by pressing *Scan for local XIMC servers*. Please note that in case of more than one local server automatic scan will pick a random one and it will require several attempts to find them all.

Warning: If both *Probe devices* and *Enumerate non-XIMC devices* options are enabled, on startup XILab will send data to all COM-ports. If the PC has multiple Bluetooth COM-ports, due to the nature of Bluetooth operation, the queries will be conducted sequentially, and connection attempts may take from a few to tens of seconds each.

The *Virtual devices* field contains the number of virtual controllers which will appear in the list of available controllers after you press the *Rescan* button or restart the XILab.

Note: *Note:* Since the libximc library opens XIMC devices in the exclusive access mode, when you start subsequent copies of XILab application, only free controllers will be found and available for selection.

User management panel provides Access Control List management for local and remote servers. This feature enables the end user to selectively grant permisions to connect and control remote 8SMC4/5-USB devices. In order to grant permission you need to create the same user with the same password locally and remotely. Deleting the user (either locally or remotely) revokes permission. By default all 8SMC4-USB-Eth1 adapters, SDK libraries and XiLab have root user preinstalled with default password.

Note: *Note:* You should have compatible Standa ethernet adapter device with ACL support to exploit this feature.

For more info please contact our technical support.

5.2.2 XILab Main window in single-axis control mode

- *Motion Control Unit*
  - *Movement without specifying the final position*
  - *Movement to the target point*
  - *Target position for motion commands*
- *Attenuator Control Unit*
- *Controller and motor status*
  - *Controller Power Supply*
  - *Motor status*
  - *Program status*
- *Group of application control buttons*
- *Status bar*

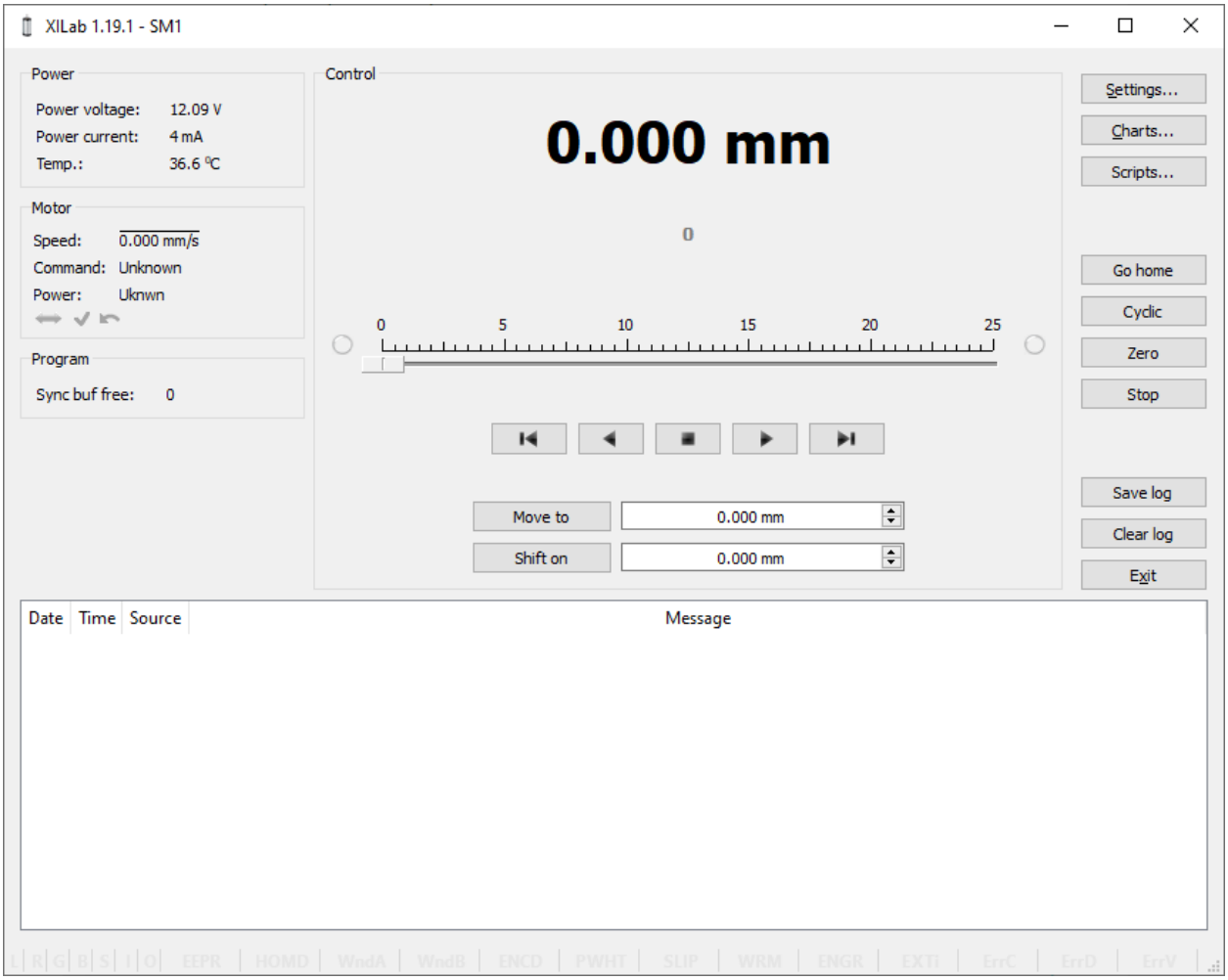


Fig. 5.4: XILab Main Window in General Motor mode

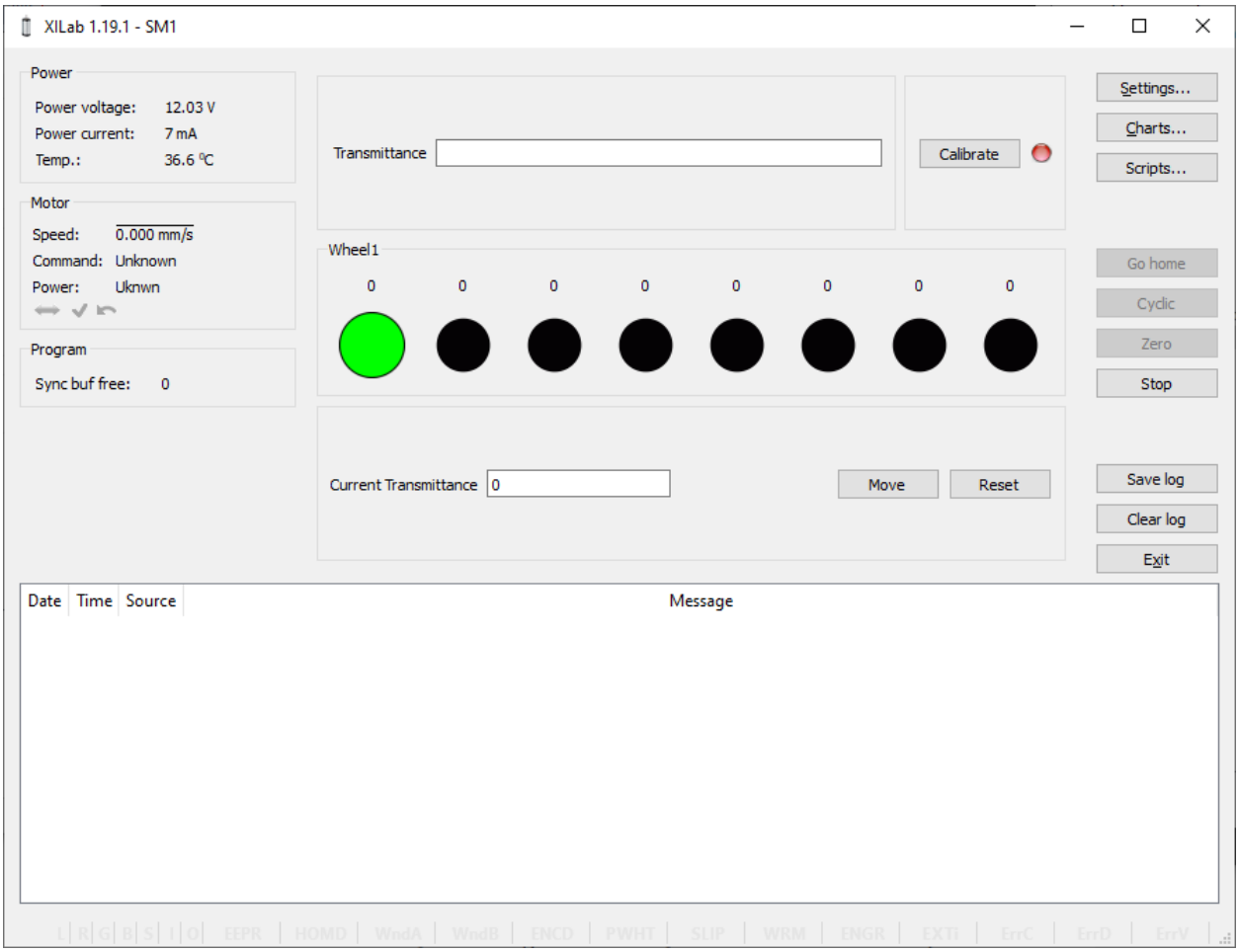


Fig. 5.5: XILab Main Window in Attenuator mode

In the left part of the window in **Power** and **Motor** groups of parameters status of the controller and the motor is available. In the central part of the window there is the **Control** group, containing the elements of motor motion control. On the right there is a group of buttons to control the application as a whole. At the bottom there is a *log*, which is hidden as the window is resized to its minimum size and a status bar. Below we consider these groups in more detail.

5.2.2.1 Motion Control Unit

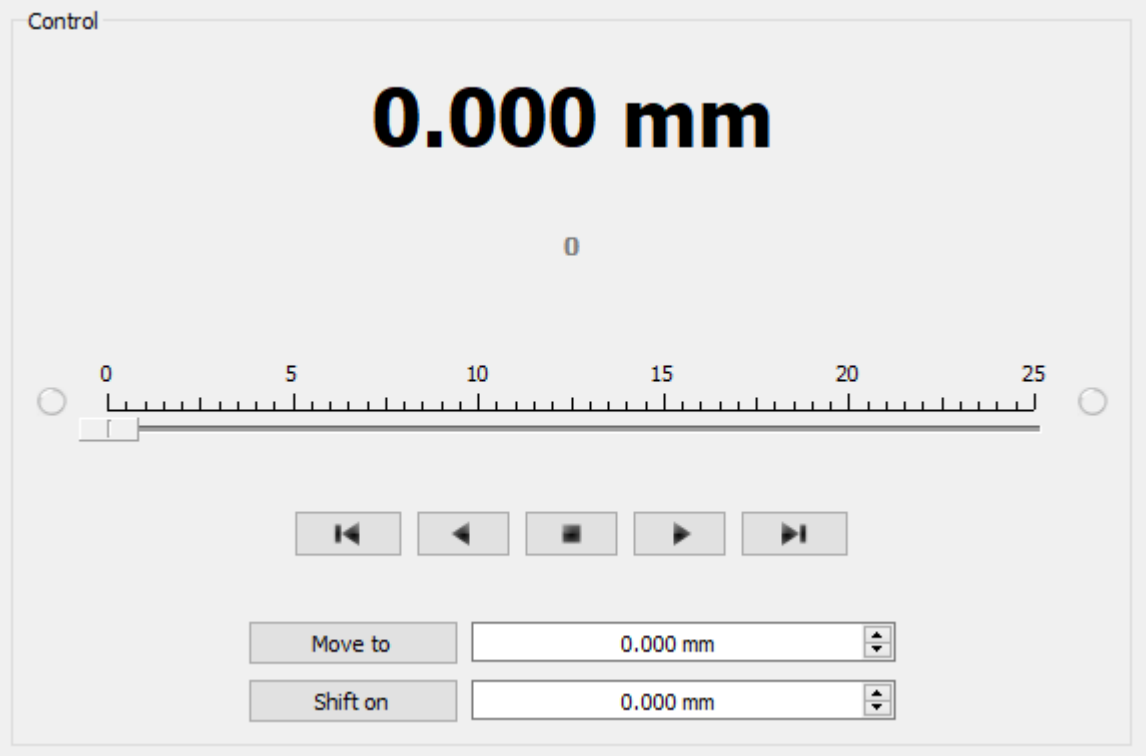


Fig. 5.6: Control Unit

In the central part of the block there is an indicator of the current position. Below it, in case the encoder is enabled, is located an encoder position indicator. In the closed loop mode (see *Operation with encoders* section) the main and the secondary indicators swap their places.

Below is the **Control** unit, containing the elements of motor motion control. Let us examine them in greater detail:

5.2.2.1.1 Movement without specifying the final position



Fig. 5.7: Movement control buttons

- The buttons *Left*, *Stop* and *Right* trigger movement to the left without specifying the final position, *stop with deceleration* any previously started movement, and start the movement to the right without specifying the final position, respectively.
- Button *Left to the border* will make the motor rotate to the left border of the slider. *Right to the border*, respectively, will do it to the right edge of the slider.
- When you press and hold the keyboard buttons *Right* or *Left* and the slider block has input focus, the movement starts in the direction of increasing or decreasing coordinate. When you release the button the movement stop as if the *Stop* button on the main window have been pressed.

5.2.2.1.2 Movement to the target point

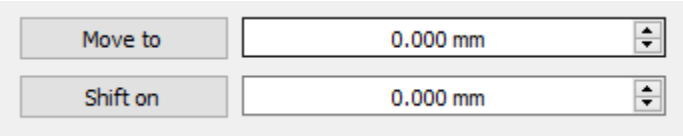


Fig. 5.8: Movement control to the given point

- *Move to* button starts the process of moving to the given position.
- *Shift on* button starts the process of shift to a given distance from the target position.

5.2.2.1.3 Target position for motion commands

Commands *Move to* and *Shift on* use the target position to calculate the movement. The target position is changed by the following commands:

Move to <value>
Target position = <value>

Shift on <offset>
Target position = target position + <offset>

Zero (provided there is no movement at the moment of sending the command)
Target position = 0

Commands *Stop*, *Left*, *Right*, *Left up to the border* and *Right up to the border* do not alter the target position.

5.2.2.2 Attenuator Control Unit

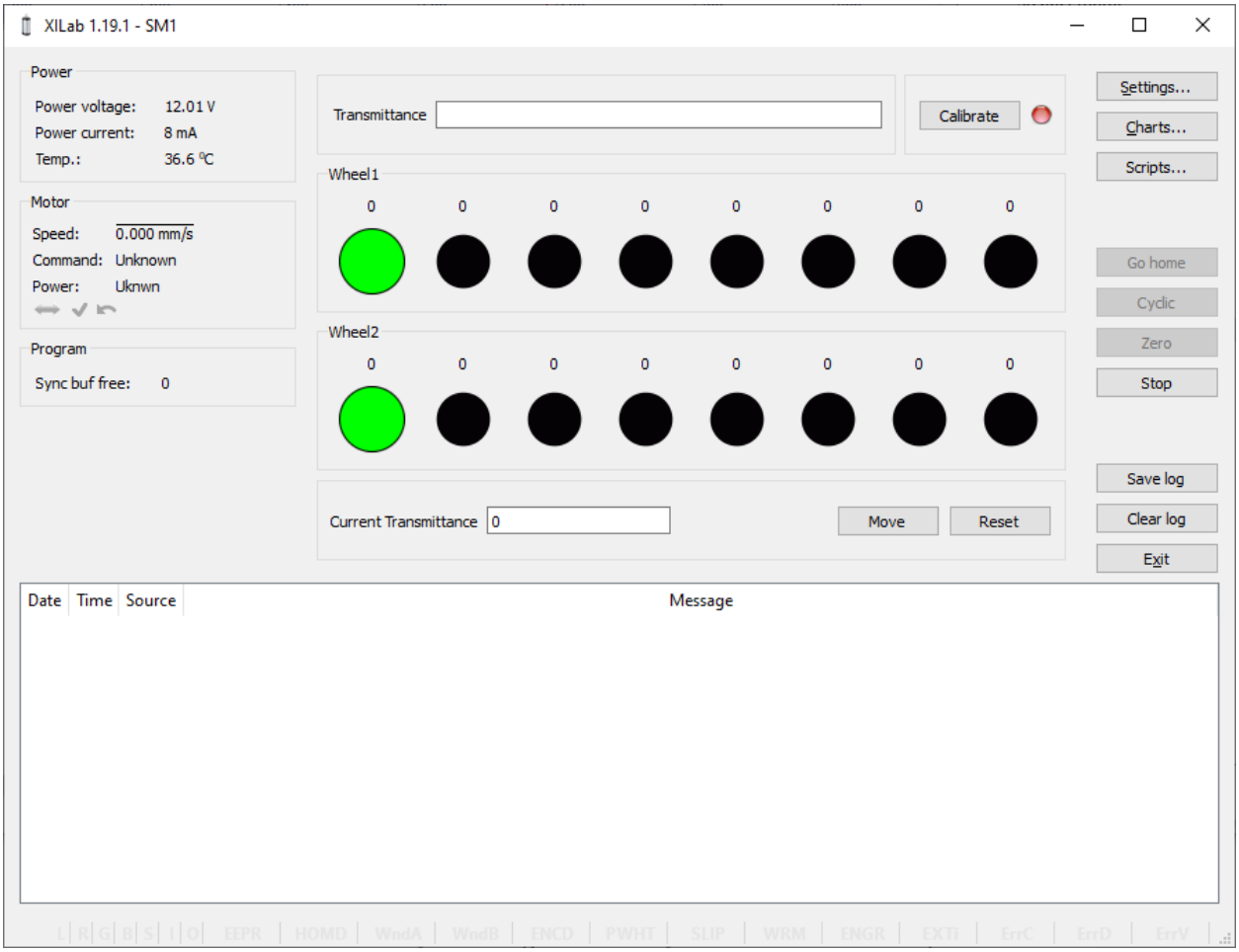


Fig. 5.9: Attenuator Control Unit

At the top of the attenuator control block a *Transmittance* window and a *Calibrate* button are located. *Transmittance* window contains a control to select desired transmittance coefficient. *Calibrate* button does manual calibration, or location of the initial position, of the attenuator - first it initiates one revolution of attenuator with current controller settings to determine relative position of attenuator wheels and then does *Automatic Home position calibration*. * Calibrate* button is optional - if the attenuator did not perform the calibration or it was reset, for example by pressing Cancel during movement, calibration will be performed before the next movement automatically.

Attenuator has one or two wheels, each wheel contains 8 optical filters. Below these filters are visualized as bars with circles. Below a *Current Transmittance* window is located. This window contains trasmittance coefficient which is the closest to desired one that can be achieved with the current filters.

Move button performs movement toward the filters corresponding to the *Current Transmittance*, that is, the highlighted ones.

Reset button deactivates all filters.

5.2.2.3 Controller and motor status



5.2.2.3.1 Controller Power Supply

Power group of parameters contains the following indicators:

- *Power voltage* - voltage supplied to the power module.
- *Power current* - current consumption of the power module.
- *Temp.* - Temperature of the controller processor.

If the color of the indicator *Power voltage* changes to red, it shows that voltage power supply exceeds the allowed value range over the acceptable value. In this case, the controller switches to *Alarm* state. You can change this parameter in the section *Critical board ratings*.

A horizontal bar above the field *Power voltage* indicates that the power voltage exceeds the motor max voltage, you can change this parameter in the section *Settings of kinematics (DC motor)*.

If color of the indicator *Power current* turns red, it shows that the current consumed by the controller from the power supply is over the acceptable value. In this case, the controller switches to *Alarm* state. You can change this parameter in the section *Critical board ratings*.

A horizontal bar above the field *Power current* indicates that the power current exceeds the motor max current, you can change this parameter in the section *Settings of kinematics (DC motor)*.

If the color of the indicator *Temp* turns red, it shows that the temperature of the controller board exceeds the acceptable value. In this case, the controller switches to *Alarm* state. You can change this parameter in the section *Critical board ratings*.

Important: It is possible to quit the *Alarm* state after terminating of the events that caused Alarm, provided that the flag *Sticky Alarm* is not set. If the flag *Sticky Alarm* is on, use the *Stop* button to quit the Alarm state.

5.2.2.3.2 Motor status

Motor group of parameters contains the following indicators:

- *Speed* - rotation speed of the motor.

- *Command* - the last performing (bold font) or executed (plain font) *controller command*. *Controller command* appears in black if the flag of the motion error MVCMD_ERROR is not set, in red otherwise. Can be one of the following options:
  - *Move to position* - move to the set position
  - *Shift on offset* - offset for a predetermined distance
  - *Move left* - move left
  - *Move right* - move right
  - *Stop* - stop
  - *Homing* - find the home position
  - *Loft* - backlash compensation
  - *Soft stop* - smooth stop
  - *Unknown* - unknown command (it may appear immediately after the controller starts)
- *Power* - state of stepper motor power supply. Can be one of the following options:
  - *Off* - motor winding is disconnected and not controlled by the driver,
  - *Short* - winding is short-circuited through the driver,
  - *Norm* - winding is powered with nominal current,
  - *Reduc* - winding is deliberately powered with reduced current relatively to operational one to reduce the power consumption,
  - *Max* - winding is powered with maximum available current, which a scheme with a given voltage supply can output.

Note: You can use the GPIO flag to detect the connected stage

A horizontal bar above the Speed parameter indicates that the speed has reached the maximum speed, which is defined in the *Settings of kinematics (DC motor)*.

5.2.2.3.3 Program status

Program group of parameters contains the following indicators:

- *Sync buf free* - free slots in the syncin command buffer (see description of ASIA command).

5.2.2.4 Group of application control buttons

- *Settings...* button opens controller settings, see *Application settings*.
- *Chart...* button opens a window with charts, see *Charts*.
- *Scripting...* button opens scripting window, see *Scripts*.
- *Go home* button searches for the home position, see *Home position settings*.
- *Cyclic* button turns on the cyclic motion, see *Cyclical motion settings*.

Note: The *Cyclic command* is a constituent command: when invoking *Cyclic* in XiLab at the controller level the sequence of *Move to* commands is executed.

- *Zero* button resets the current position of the motor and the encoder value.
- *Stop* button sends the command of *immediate stop*, resets the Alarm state, clears the command buffer for synchronous motion and stops a script if it is running.
- *Save log* button saves the contents of the log to a file in CSV format (opens a file selection dialog window).
- *Clear log* button clears the contents of the log.
- *Exit* button performs safe shutdown, see *Correct shutdown*.

5.2.2.5 Status bar

Status bar contains current controller status indicators. From left to right these are 7 flags,

- **L** - Left button state
- **R** - Right button state
- **G** - State of external GPIO pin
- **B** - State of brake pin
- **S** - State or revolution sensor pin
- **I** - State of sync in pin
- **O** - State of sync out pin

and separate indicators (flags)

- **EEPR** - Lights up green when the stage is equipped with EEPROM memory chip. The built-in stage profile is uploaded from the EEPROM memory chip if the EEPROM_PRECEDENCE flag (XILab Settings->About device tab) is enabled, allowing you to connect various stages to the controller with automatic setup.
- **HOMD** - Lights up after successful execution of home() command meaning that relative position scale is calibrated against a hardware absolute position sensor like a limit switch. Drops after loss of calibration like harsh stop and possibly skipped steps.

- **WndA/WndB** has 1 of 4 statuses: - Winding A/B is disconnected. - Winding A/B state is unknown. - Winding A/B is short-circuited. - Winding A/B is connected and working properly.

Important: Status is determined using statistical data while moving, taking its time, and turning this status rather useless in common applications. Therefore this function have been disabled for the moment.

- **ENCD** - Encoder state has 1 of 5 statuses: - Encoder is absent. - Encoder state is unknown. - Encoder is connected and malfunctioning. - Encoder is connected and operational but counts in other direction. - Encoder is connected and working properly.
- **PWHT** - Power driver overheat. Motor control is disabled until some cooldown. This should not happen in boxed versions of controller. This may happen in bare board version of controller with a custom radiator. Redesign your radiator then.
- **SLIP** - Motor slip detected. Flag is set when encoder position and step position are too far apart. You can disable the "Position control" flag or increase the error in the "threshold" field on the XILab Settings->Position control tab to stop these error from happening.

- **WRM** - Lights up when there is a substantial difference between stepper motor windings resistances. This usually happens with a damaged stepper motor with partially short-circuited windings. You can diagnose the problem according to the instructions *in our manual*

Important: The WRM algorithm was not originally designed to be used for belt-driven stages (for example, 8MRB450-350 or 8MRB240-152) due to the fact that the belt can stretch and vibrate. Vibration usually occurs at high speeds, which knocks down the work of the WRM algorithm. For belt drive stages, this is normal behavior

---

- **ENGR** - Lights up red when the motor control error occurs. Motor control algorithm failure means that it can't define the correct decisions with the feedback data it receives. Single failure may be caused by mechanical problem. A repeating failure can be caused by incorrect motor settings.
- **EXTI** - The error is caused by the external EXTIO input signal set to cause this error in the settings (XILab Settings->EXTIO tab)
- **ErrC** - Command error encountered. The command received is not in the list of controller known commands. Most possible reason is the outdated firmware that can be updated in XILab Settings->About device tab->Autoupdate button.
- **ErrD** - Data integrity error encountered. The data inside command and its CRC code do not correspond, therefore data can't be considered valid. This error may be caused by EMI in UART/RS232 interface.
- **ErrV** - Value error encountered. The values in the command can't be applied without correction because they fall out the valid range. Corrected values were used instead of the orginal ones.

5.2.3 XILab Main window in multi-axis control mode

Important: XILab allows you to open up to 32 axes at the same time, but only the first 3 (X/Y/Z axes) will be displayed in the multi-axis interface. However, it is possible to work with other open axes (those that are not visually displayed in XILab), for this you need to use the XILab scripting language. To visually work with a large number of axes (more than three), you will need to launch several XILab windows. For example, you have a 5-axis controller, then in the first XILab window you can open the first three axes, and in the second window the remaining two.

If you need to open more than 3 axes in one window and visually control them, you can use unsupported examples written by *in C* and *Python*

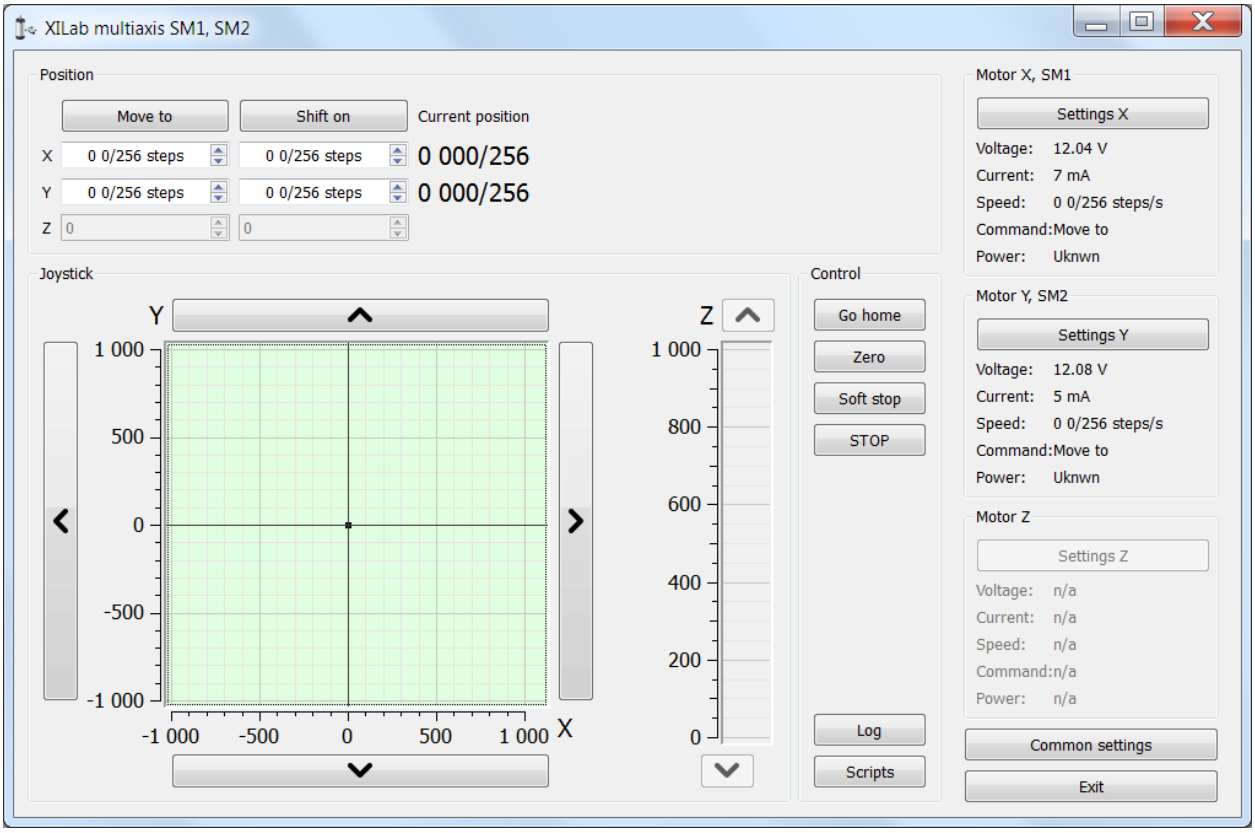


Fig. 5.10: XILab Main window

In the top left part of the screen in the **Position** parameter group there are indicators of the current position. In the bottom left part of the window there is the **Joystick** and **Control** blocks, which are a graphical control element for several axes and a button block respectively. In the top right part, in the **Motor** blocks data on the current status of controllers and connected motors is located. In the bottom right part of the window there is a group of buttons for application control as a whole. Let us consider these groups in more detail.

5.2.3.1 Motion control block

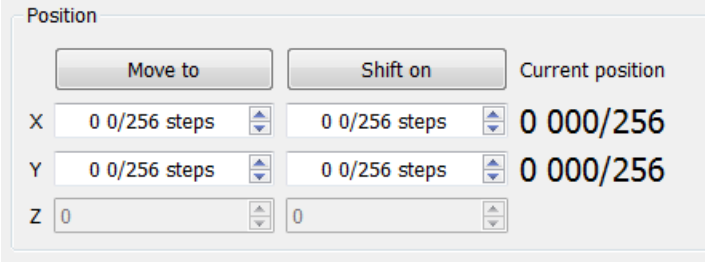


Fig. 5.11: Motion control block

In the *Current position* column indicators of the current position in steps or calibrated units (see below) for the axes X, Y and Z (from left to right) are located. The *Move to* button performs movement to the coordinate given by the controls of its column, and the *Shift on* button performs shift on a specified distance from the current position. If one of the controllers is temporarily absent or disabled, the corresponding line becomes grayed out.

5.2.3.2 Virtual joystick block

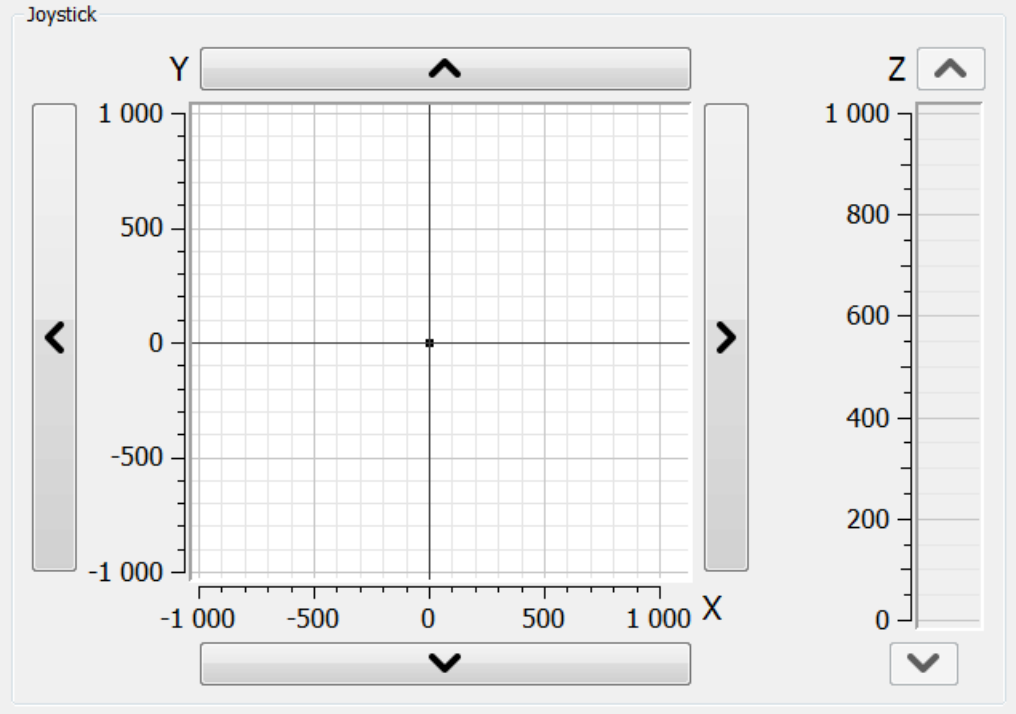


Fig. 5.12: Virtual joystick block

In this block, the current coordinate of the controllers is visualized as a dot with two lines on the plane for X-Y axes and the line for the Z axis.

There are several possible ways to control movement of the controllers:

- When you click anywhere on the X-Y plane or in the Z column, the corresponding controller or controllers start to move to the selected coordinate in accordance to its own movement settings.
- When you press and hold the screen buttons with up, down, left and right arrows, the corresponding axis starts to move in that direction. The movement *stops with deceleration* when you release the button (soft stop command is sent).
- When you press and hold the keyboard buttons Right, Left, Up, Down, PageUp or PageDown when the joystick block has input focus, the axis X, Y or Z, respectively, starts to move in the direction of increasing or decreasing coordinate. The movement *stops with deceleration* when you release the button (soft stop command is sent). On receiving input focus the widget background color changes from white to light-green.

The scales of the axes are set in *Slider settings* block on *General motor* tab in Settings window separately for each controller. If *user units* are being used, then the corresponding axis scale is measured in these units. If the position read from any controller is out of its axis range, then the corresponding indicator will not appear on the screen.

5.2.3.3 Control block



Fig. 5.13: Control block

The *Go home* button searches for the home position independently for each of the controllers; see *Home position settings*.

The *Zero* button resets the current position of the motor and value of the encoder for each controller.

The *Soft stop* button executes a soft stop for each of the controllers.

Note: The *STOP* button sends an *immediate stop* command to each controller, resets their Alarm statuses, clears their command buffers for synchronous motion and stops a script if one is running.

The *Log* button opens a window with log information. Here you can find diagnostic information (errors, warnings, informational messages) from XiLab, libximc and script sources.

The *Scripts* button opens a scripting window, see *Scripts*.

5.2.3.4 Block of status indicators for controllers and motors

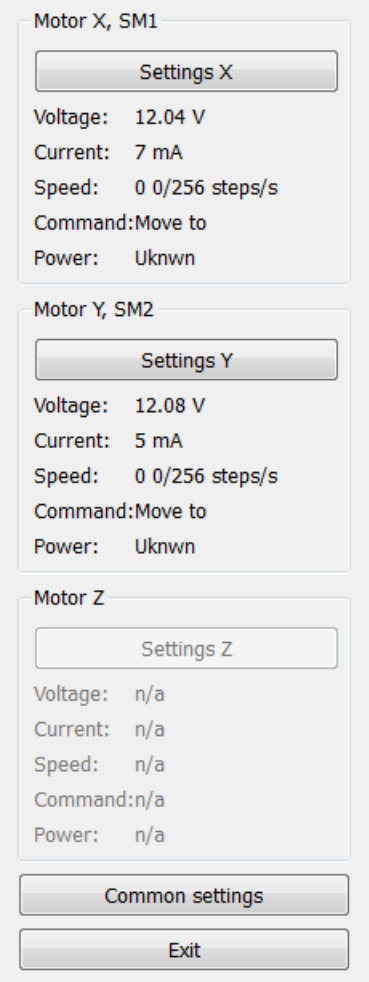


Fig. 5.14: Block of status indicators for controllers and motors

Here are the three blocks of indicators of controllers and motors for axes X, Y and Z. In the title of the block the serial number of corresponding controller is displayed. Each block contains the following indicators:

- Voltage - the voltage at the power section of the motor.
- Current - current power consumption of the motor.
- Speed - current speed of the motor.
- Command - the last command of the controller that was executed, or is being executed.
- Power - the state of the motor power supply.

The buttons Settings X, Y, Z open the configurations of a controller, which corresponds to this axis, see *Application settings*.

Below the indicator blocks *Common settings* and *Exit* buttons are located.

The *Common settings* button opens a dialog window with general settings. Here you can choose which controller serial numbers are considered “X”, “Y” and “Z” axes, also this window contains logging settings.

The *Exit* button performs proper shutdown, see *Correct shutdown*.

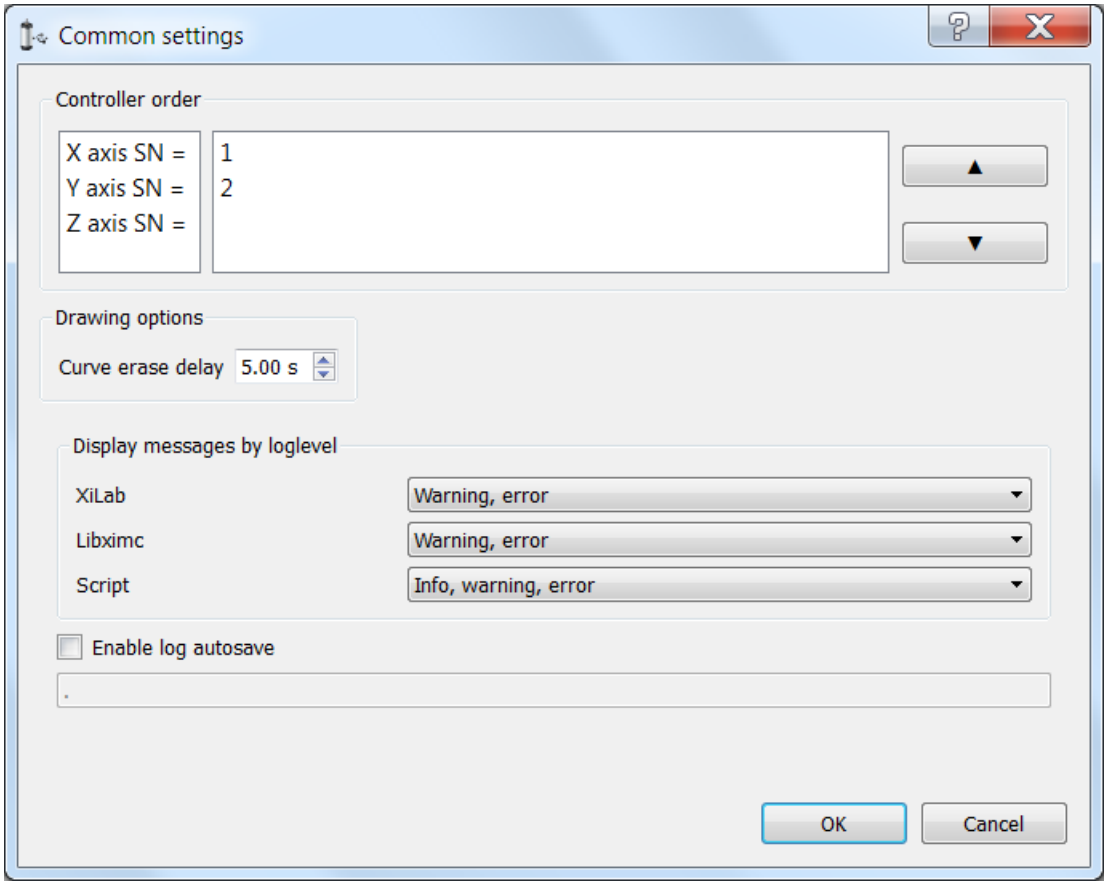


Fig. 5.15: Common settings dialog window

Common settings window contains axis order settings, curve drawing options and logging settings.

You can reorder axes by choosing the axis serial, then pressing “up” or “down” buttons on the right.

First axis in the list, that is, an axis with serial number listed to the right of “X axis SN = ” label will be referred to as “X axis”, the second one as “Y axis”, the third one, if it is available, as “Z axis”.

If you enter a greater than zero value of N seconds in the “curve erase delay” control in the “Drawing options”, then last N seconds of X and Y axis controller motion will be displayed as a trajectory in 2D-space overlaid on the virtual joystick window.

Common logging settings are completely identical to *single-axis version logging settings*.

Here you can configure logging detail (“None” for no logging, “Error” to only log errors, “Error, Warning” to log errors and warnings, “Error, Warning, Info” to log errors, warnings and informational messages) for each source: XiLab itself, libximc library and Scripts module.

If the autosave option is enabled the log is saved to the file, which is flushed every 5 seconds. It has a name of type “xilab_log_YYYY.MM.DD.csv”, where YYYY, MM and DD are current year, month and day respectively. The log is saved in CSV format.

5.2.4 Application settings

Settings button from the *main window* opens the Settings window.

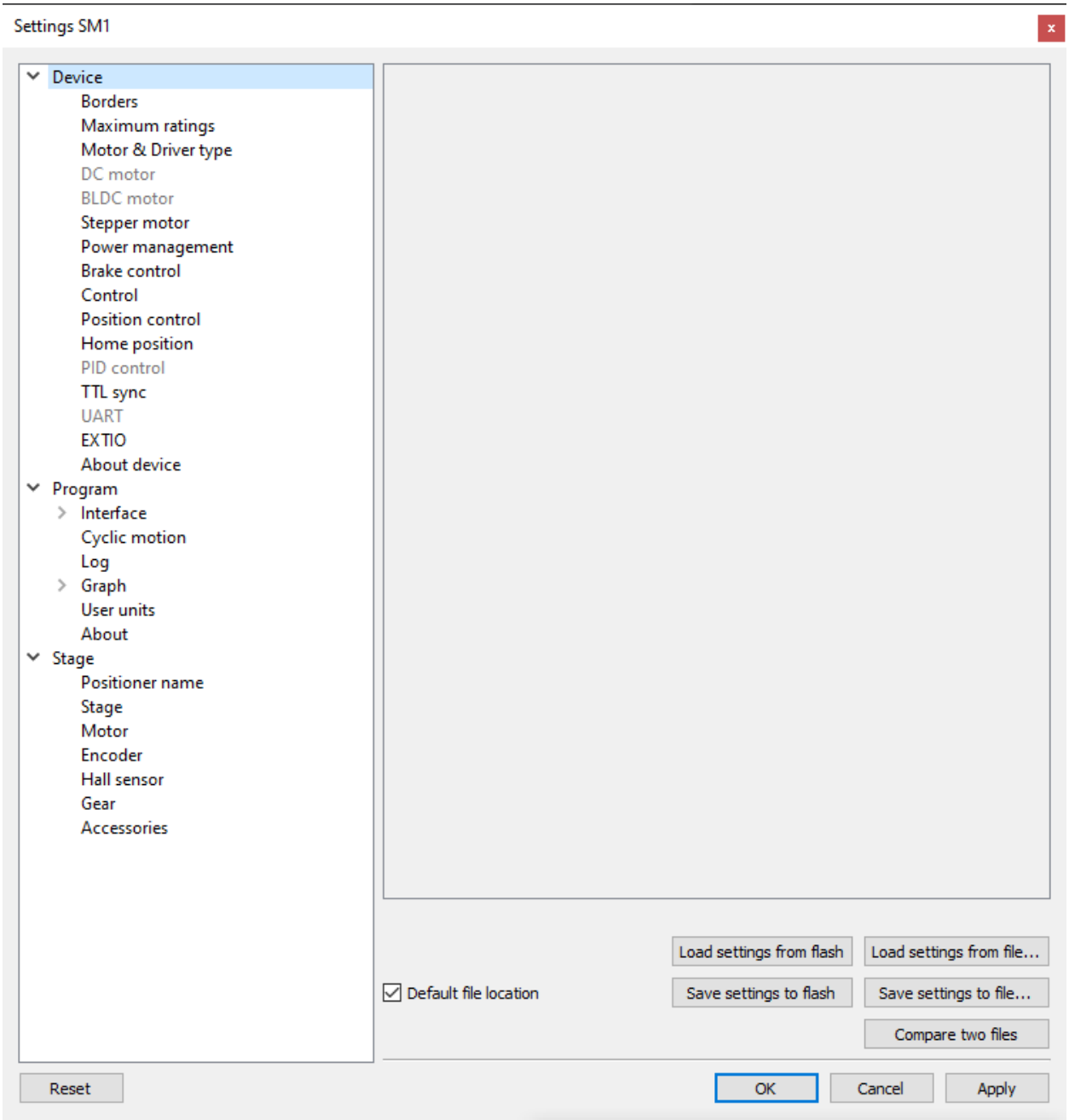


Fig. 5.16: XILab Settings Main Window

Application settings are presented as a hierarchical tree and are divided into three groups: controller settings - “Device”, XILab application settings - “Program”, characteristics of a stage - “Stage”.

The first group *Device* contains the parameters that can be stored directly in the device (in the flash memory or in the RAM of controller).

The second group *Program* contains the XILab application settings, which are not written into the controller, and are used to control the XILab itself.

The third group *Stage* contains information about the parameters of the stage and is read from its ROM chip.

Important: The information on the *"Stage"* tab temporarily not used

Description of **Load setting from flash** and **Save settings to flash** buttons is located in the *Saving the parameters in the controller flash memory*.

All application settings from the first two groups can be saved to an external file when you click **Save settings to file**.

When you click the **Load setting from file...** button you can pick a file with application settings to be loaded into Settings window.

When you click the **Compare two files** button, a dialog window with file selection is opened. If you select two files all their settings are compared and a list of differences is displayed. Missing keys in one of the files are marked in the table as "<NO KEY>".

The **OK** Button closes the Settings window and saves all changes to the settings to the controller. The **Cancel** button closes the window without saving. The **Apply** button saves the settings without closing the window.

The **Reset** button resets all setting changes that were made since the **Apply** button was pressed, or after opening the Settings, if the **Apply** button was not pressed.

Note: Only Device configuration settings can be saved into the flash memory of the controller.

5.2.5 Charts

Button **Chart** of the *main window* opens a window for working with charts.

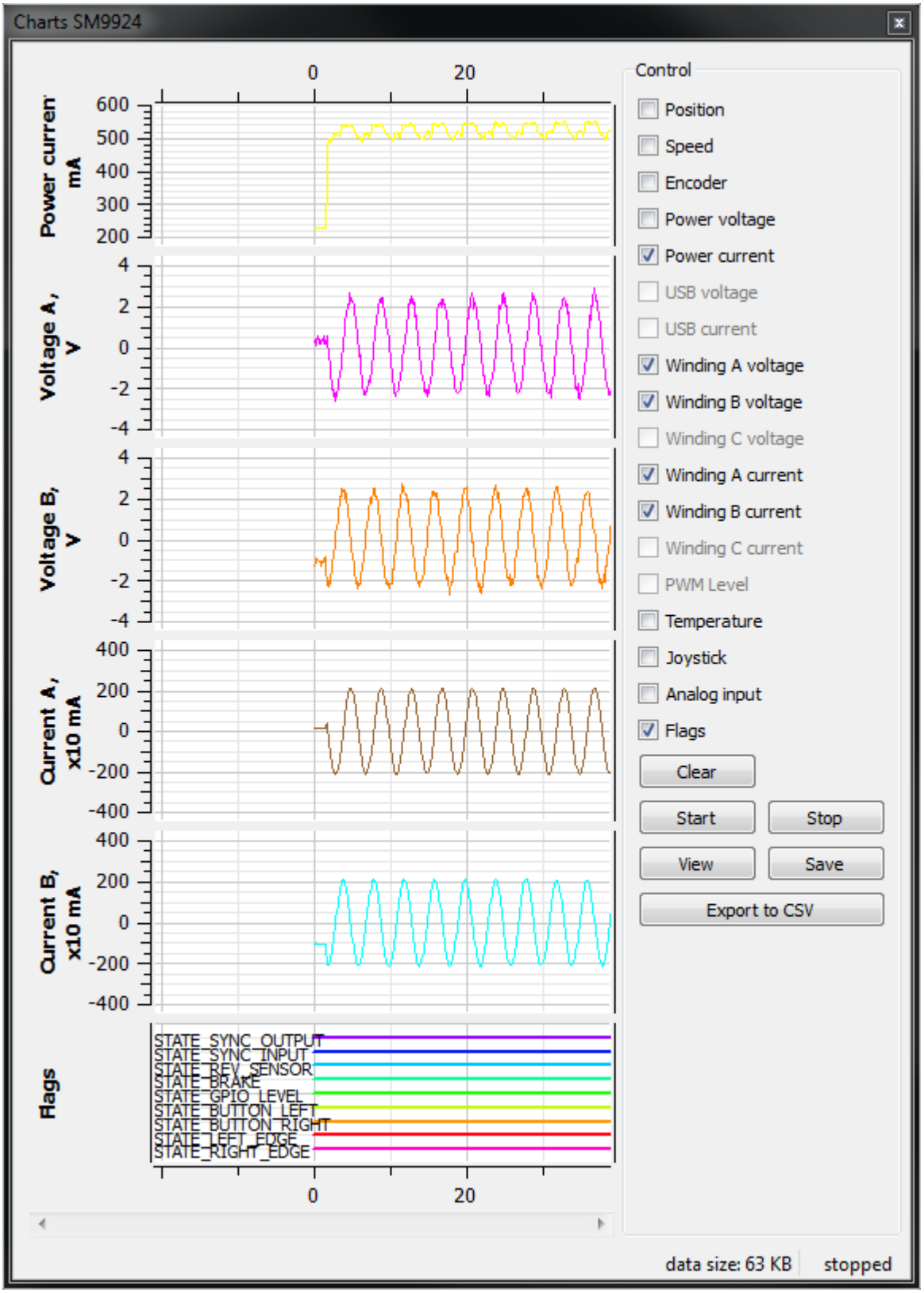


Fig. 5.17: XILab Charts window

In the left part of the window there are Charts of variables, in the right side of the window there is the **Control** block, which contains the charts control elements. Above the **Control** block there are flags which enable different charts, below the **Control** block there is a group of buttons to control the stored charts data.

5.2.5.1 Values displayed on the charts

- *Position* is the primary field in which the current position is stored, no matter how feedback is arranged. In the case of a DC-motor this field has the current position according to the encoder, in the case of a SM (stepping motor) this field contains the current position in steps;
- *Speed* is the current speed;
- *Encoder* is the position according to the secondary position sensor;
- *Power voltage* is voltage of the power section;
- *Power current* is current consumption of the power section;
- *USB voltage* is voltage on the USB;
- *USB current* is current consumption by USB;
- *Winding A current* - in the case of stepper motor, the current in winding A; in the case of a brushless motor, the current in the first winding and in the case of DC motor, the current in its only winding;
- *Winding B current* - in the case of stepper motor, the current in the winding B; in the case of brushless motor, the current in the second winding, unused in the case of DC motor;
- *Winding C current* - in the case of a brushless motor, the current in the third winding, unused in the case of DC or stepper motor;
- *Winding A voltage* - in the case of stepper motor, the voltage on winding A; in the case of a brushless motor, the voltage on the first winding and in the case of DC motor, the voltage on its the only winding;
- *Winding B voltage* - in the case of stepper motor, the voltage on the winding B; in the case of brushless motor, the voltage on the second winding, unused in the case of DC motor;
- *Winding C voltage* - in the case of a brushless motor, the voltage on the third winding, unused in the case of DC or stepper motor;
- *PWM level* the PWM duty factor (only for DC motors);
- *Temperature* is temperature of controller processor;
- *Joystick* is value of input signal from the joystick;
- *Analog input* is value of analog input;
- *Flags* shows the state of the controller flags.

5.2.5.2 Button functions

- *Clear* - clears the stored data and the Charts window;
- *Start* - starts recording the data and displaying charts. If the option "Break data update when motor stopped" in **Program -> Graph** is turned on, no data recording and charts auto-scrolling will occur when the motor is stopped;
- *Stop* - stops data reading;
- *Save* - stores chart data to a file;
- *Load* - loads chart data from a previously saved file;
- *Export to CSV* - exports chart data to CSV file

5.2.5.3 Limit value

If a limit is set for speed, power voltage power current, this limitation is displayed on the graph in dotted lines:

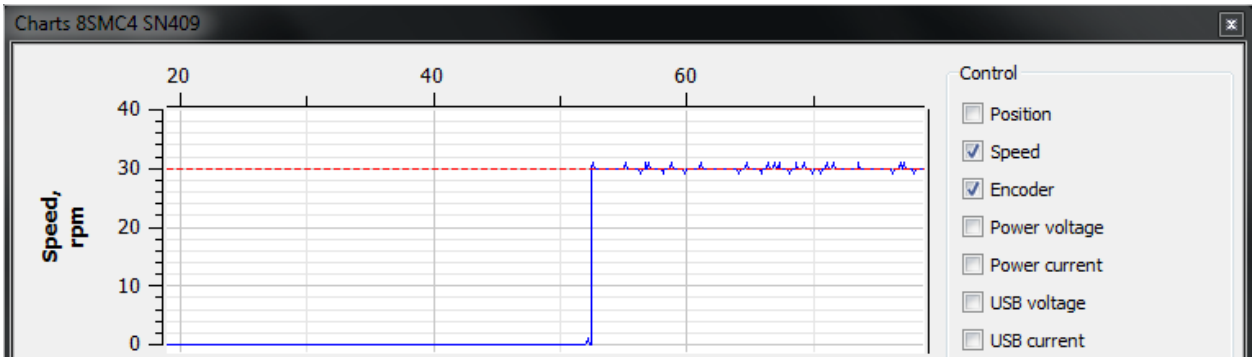


Fig. 5.18: Speed chart in XiLab program with speed limitation

5.2.6 Scripts

The “Scripts” button on the *main window* opens a window for working with scripts.

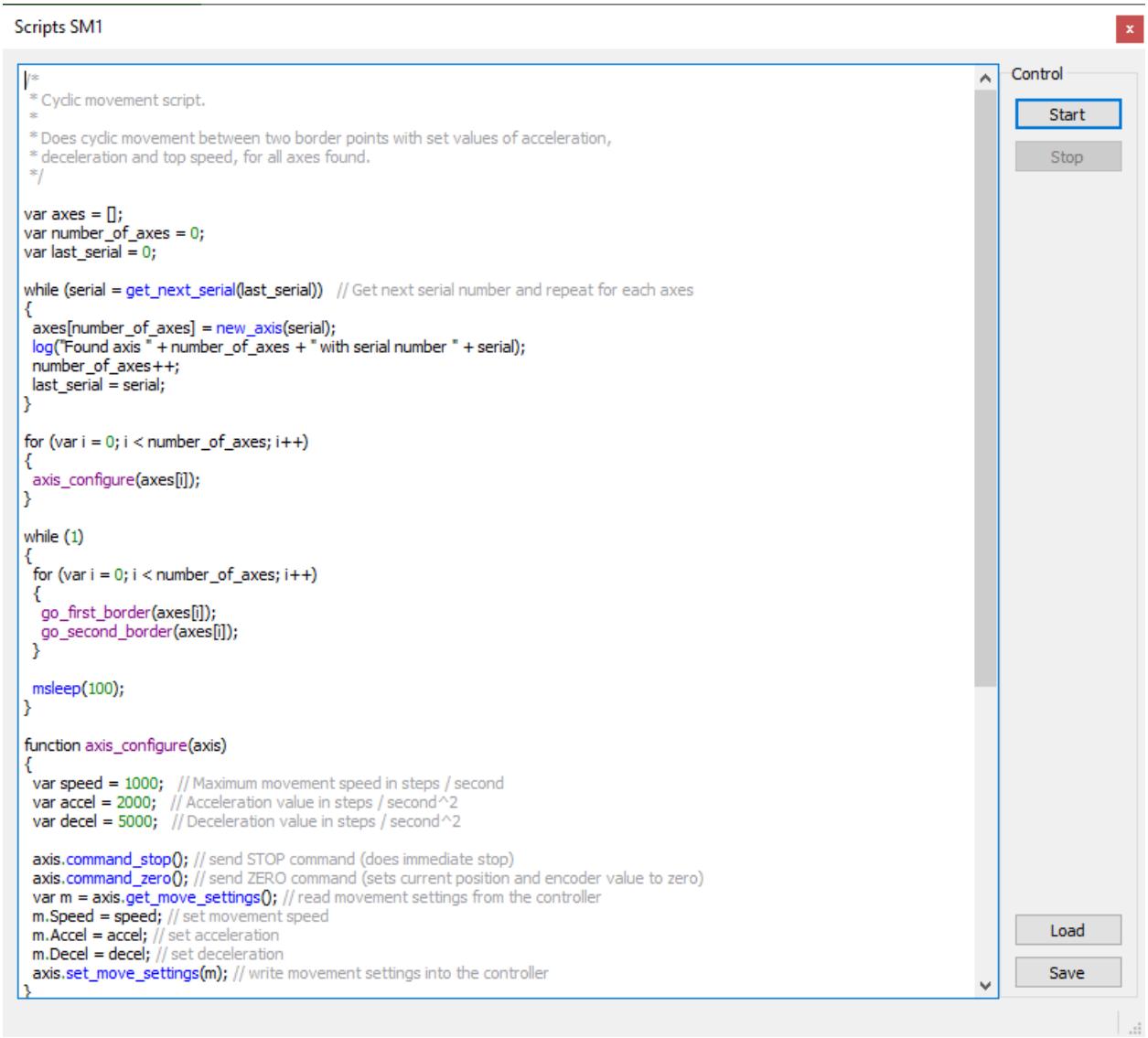


Fig. 5.19: XILab scripting window

On the left side of the window a text edit field is located, on the right side of the window a Control block is located, which contains control buttons.

5.2.6.1 Button functions

- *Start* - launches the script. The button is inactive if the script is already running. Right after the button is pressed and before the script is interpreted the body of the script is autosaved to a temporary file (see below).
- *Stop* - stops the script. Inactive if the script is not running.
- *Save* - opens a file save dialog, prompting the user pick a file to save current script text to. Inactive if the script is running.
- *Load* - opens a file load dialog, prompting the user to pick a file to load into the script window. Inactive if the script is running. Warning! If you load a file all unsaved changes will be lost.

XILab loads last saved script text into the Scripting window on startup. Autosave runs on every script start and on

XILab exit. Autosave file is named “scratch.txt” and is located in user settings directory.

Note: *Stop* button in XILab main window also stops script execution, acting as an emergency stop button.

The *STOP* button sends an *immediate stop* command to each controller, resets their Alarm statuses, clears their command buffers for synchronous motion and stops a script if one is running.

Scripting language description is located in the Programming section of the manual.

5.2.7 XILab log

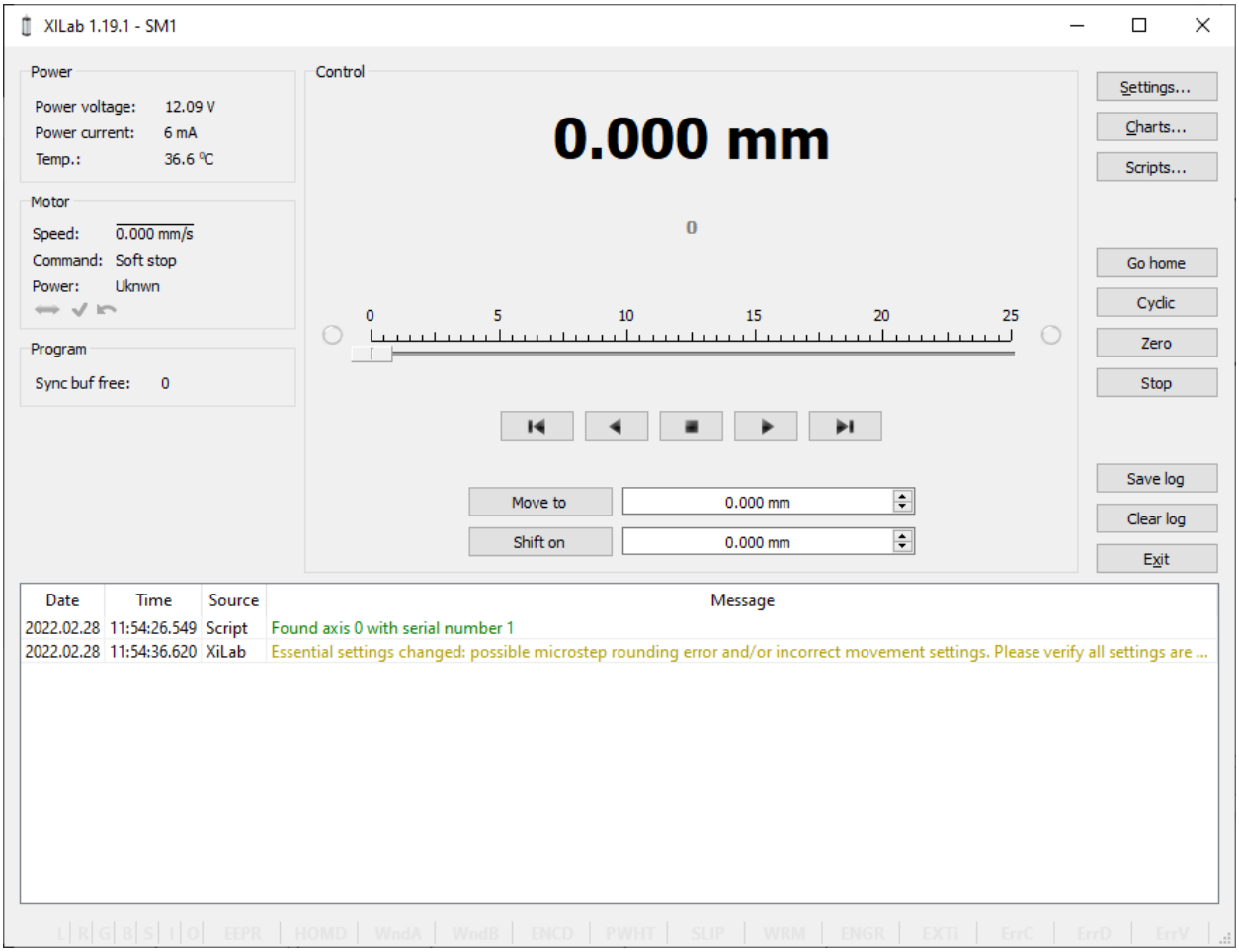


Fig. 5.20: XILab log window

XILab log at the bottom part of the main window shows libximc library messages. It also shows messages from XILab application and *Scripts* interpreter.

Log has 4 columns: date and time of record, the source and the message text.

Messages have a logging level indicating message importance: error, warning and informational message. Error messages are red, warnings are yellow and informational messages are green.

You can set a filter on displayed messages on the *Log settings* page in the Settings window.

5.3 Controller Settings

5.3.1 Settings of kinematics (stepper motor)

In the *Application settings* **Device -> Stepper motor**

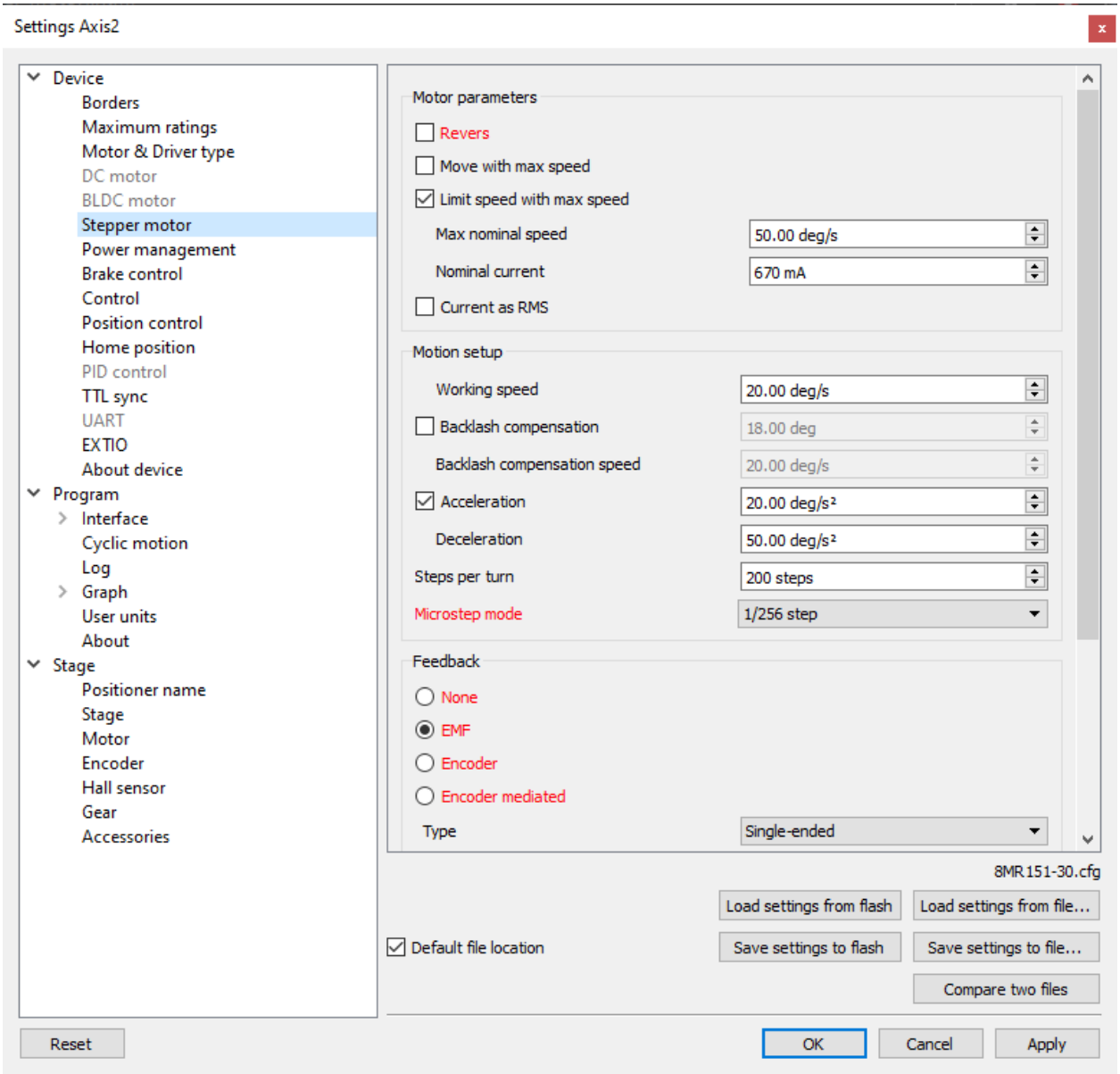


Fig. 5.21: Settings of stepper motor kinematics window

5.3.1.1 Motor parameters - directly related to the electric motor settings

Revers - checking this flag associate the motor rotation direction with the position counting direction. Change the flag if positive motor rotation decreases the value on the position counter. This flag effect is similar to the motor winding reverse polarity.

Move with max speed - if this flag is checked motor ignores the preset speed and rotates at the maximum speed limit.

Limit speed with max speed - if this flag is checked the controller limits maximum speed to the value specified in the *Max nominal speed* field. For example, if the speed exceeds the rated value, controller will reduce output action, until the speed come back to the normal range. However, the controller remains operational and will continue the current task.

Max nominal speed - motor rated speed.

Nominal current - motor rated current. The controller will limit the current with this value.

Current as RMS - if this flag is checked engine current value is interpreted as root mean square current value. If the flag is unset, then engine current value is interpreted as maximum amplitude value. See *Calculation of the nominal current* page for description

5.3.1.2 Motion setup - movement kinematics settings

Working speed - movement speed.

Backlash compensation - backlash compensation. Since the stage mechanics are not ideal there is a difference between approaching a given point from the right and from the left. When the backlash compensation mode is on the stage always approaches the point from one side. The preset value determines the number of steps which the stage takes to pass a given point in order to come back to it from the same side. If the specified number is above zero the stage always approaches the point from the right. If it is below zero the stage always approaches the point from the left.

Backlash compensation speed - speed of backlash compensation. When the backlash compensation mode *Backlash compensation* is on the stage approaches the point from the right or from the left with a preset speed determined in the number of steps per second.

Acceleration - enables the motion in acceleration mode, the numerical value of the field is the acceleration of movement.

Deceleration - movement deceleration.

Steps per turn - determines the number of steps for one complete motor revolution. The parameter is set by user.

Microstep mode - step division mode. 9 modes are available: from a whole step to the 1/256 of a step. Description of modes is in the *Supported motor types*.

5.3.1.3 Feedback settings

An encoder can be used as feedback sensor for stepper motors. Three feedback modes are available for stepper motors.

None - is without feedback. The movement is carried out in steps.

Encoder - mode of motion setting in the encoder reference values. The following encoder types are available: Single-ended, Differential or Autodetect.

Encoder mediated - in this case, the movement is carried out in several iterations with position control at the end of each iteration of the encoder.

Encoder counts per turn - this parameter defines the number of *encoder* pulses per one full motor axis revolution.

Encoder reverse - interpret encoder signal as if it were reversed.

5.3.2 Motion range and limit switches

In the *Application settings* **Device -> Borders**

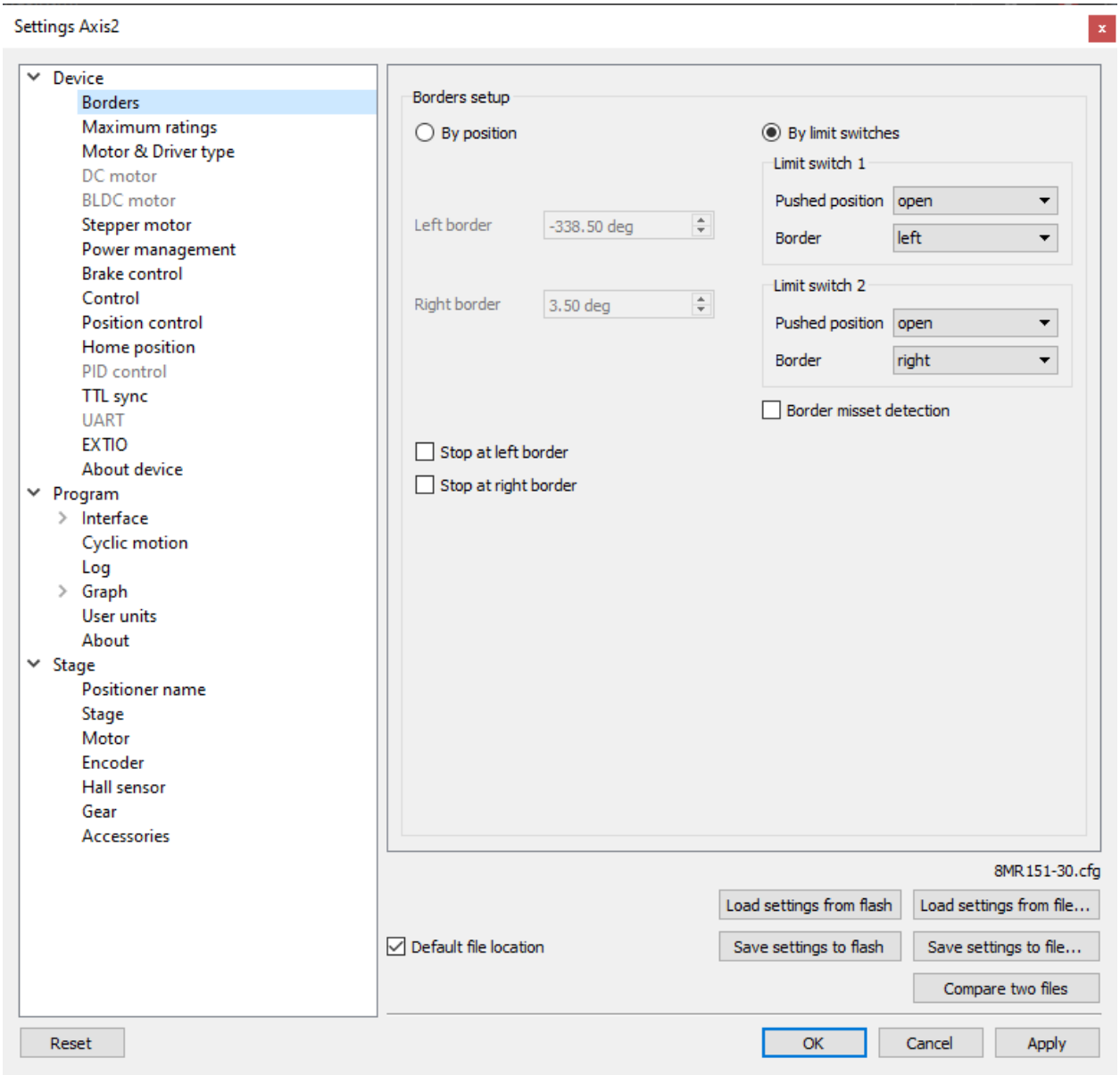


Fig. 5.22: Motion range and limit switches settings window

Borders setup parameter group contains borders and limit switches parameters. These parameters are used to keep the stage in the permissible physical movement limits or motion range limit in accordance with the user requirements. Borders can be set either by position (internal controller step counter) or by *limit switches* located in the stage terminal points.

To set the borders by position select the *By position* and specify the *Left border* and *Right border* values, which correspond to the left and right edge respectively.

To set the borders by the limit switches select *By limit switches* and set up both *Limit switch 1* and *Limit switch 2*.

Pushed position - sets the limit switch condition when it is reached: open or closed.

Border - sets the limit switch position: on the left or on the right of the stage working range.

Check the *Stop at left border* and / or *Stop at right border* for a forced stop of motor when the border is reached. In

this case the controller will ignore any commands of movement towards the limit switch if the corresponding limit switch has already been reached.

When the border position is reached the corresponding indicator flashes in the main application window.

If the *Border misset detection* flag is checked, the engine stops upon reaching of each border. This setting is required to prevent engine damage if limit switches appear to be potentially incorrectly configured. Read more about controller operation in this mode in the *limit switches location on stages*.

5.3.3 Critical board ratings

In the *Application settings* **Device -> Maximum ratings**

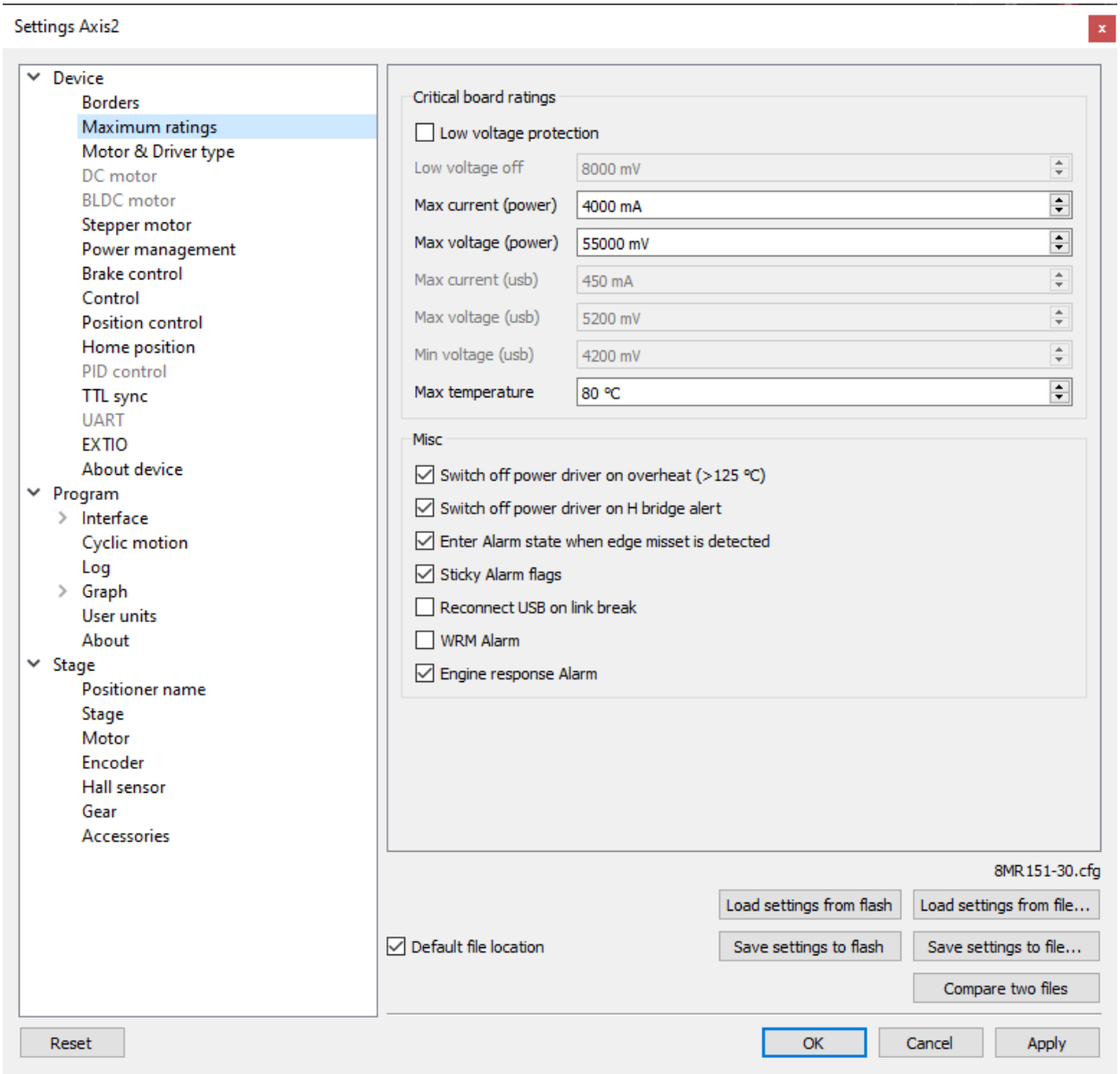


Fig. 5.23: Controller critical parameters settings window

Critical board ratings - This group is responsible for the maximum values of input current *Max current (power)* and

voltage *Max voltage (power)* on the controller, the maximum current *Max current (usb)* and voltage *Max voltage (usb)* for USB, the minimum voltage *Min voltage (usb)* for USB, and the board temperature *Temperature* (if the temperature is measured on the given controller version).

If the controller current consumption value, power supply voltage or the temperature goes out of the allowed range defined here then the controller turns off all power outputs and switches to *Alarm* state. At the same time the Alarm state information will appear in the main window (window background will change to red) and the out of range parameter value will be displayed in blue or red (below or above the limit respectively).

If the *Low voltage protection* flag is checked the low voltage supply protection is active. Then *Low voltage off* and below values of input voltage turns the controller into *Alarm* state.

The group **Misc** includes all other critical parameters settings.

Switch off power driver on overheat (> 125 ° C) - this parameter enables the *Alarm* state in case of power driver overheat.

Switch off power driver on H bridge alert - this parameter enables the *Alarm* state in case of the power driver malfunction signal.

Enter Alarm state when edge misset is detected - this parameter enables the *Alarm* state in case of incorrect boundary detection (activation of right limit switch upon moving to the left, or vice versa).

Sticky Alarm flags - locking of Alarm condition. If the *Sticky Alarm flags* check-box is unchecked the controller removes the Alarm flag as soon as it its cause is removed (e.g. in case of over-current the windings are disconnected, which results in the decreased current). If the *Sticky Alarm flags* is enabled, the Alarm mode cause and the *Alarm* mode are canceled by the *Stop* command only.

Reconnect USB on link break - if this option is enabled, then the controller will reset USB link if the connection breaks.

5.3.4 Power consumption settings

In the *Application settings* **Device -> Power Management**

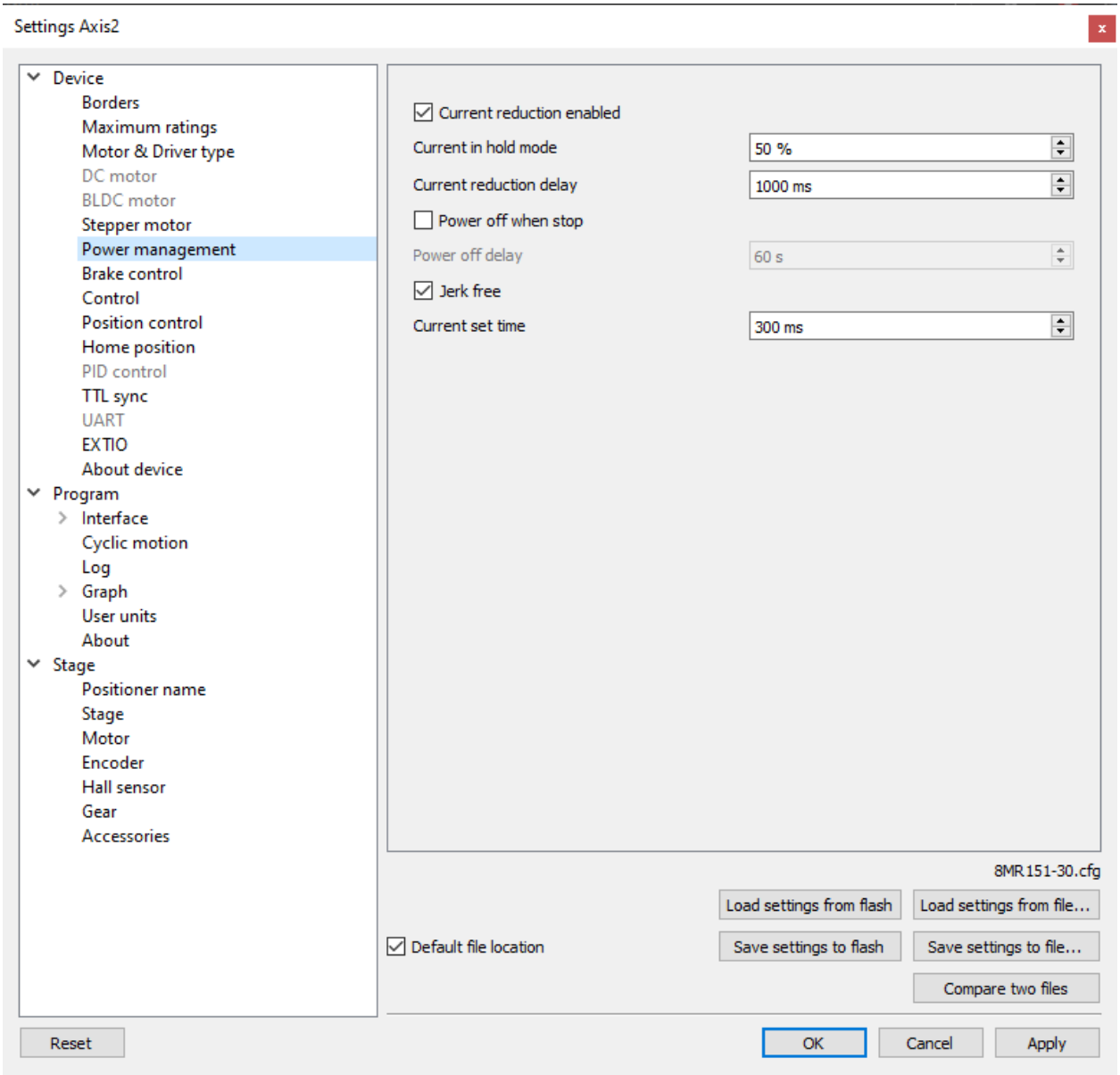


Fig. 5.24: Power consumption settings window

Current reduction enabled - activates the reduced energy consumption mode.

- *Current in hold mode* - it determines the current in the hold mode in % of the nominal value. Value range: 0 .. 100%.
- *Current reduction delay* - parameter determines the delay between switching to the STOP mode and power reduction activation. It is measured in milliseconds. Value range: 0 .. 65535 ms.
- *Power off when stop* - it activates the function that deenergized the motor windings after switching to the STOP state.
- *Power off delay* - parameter determines the delay in seconds between switching to the STOP mode and motor power-off. Value range: 0 .. 65535.
- *Jerk free* - activates the current smoothing function to eliminate the motor vibration.

- *Current set time* - parameter determines the time for jerk free current setting in milliseconds. Value range: 0 .. 65535 ms.

Detailed description of these parameters see in the *Power control* section.

5.3.5 Home position settings

In the *Application settings* **Device -> Home position**

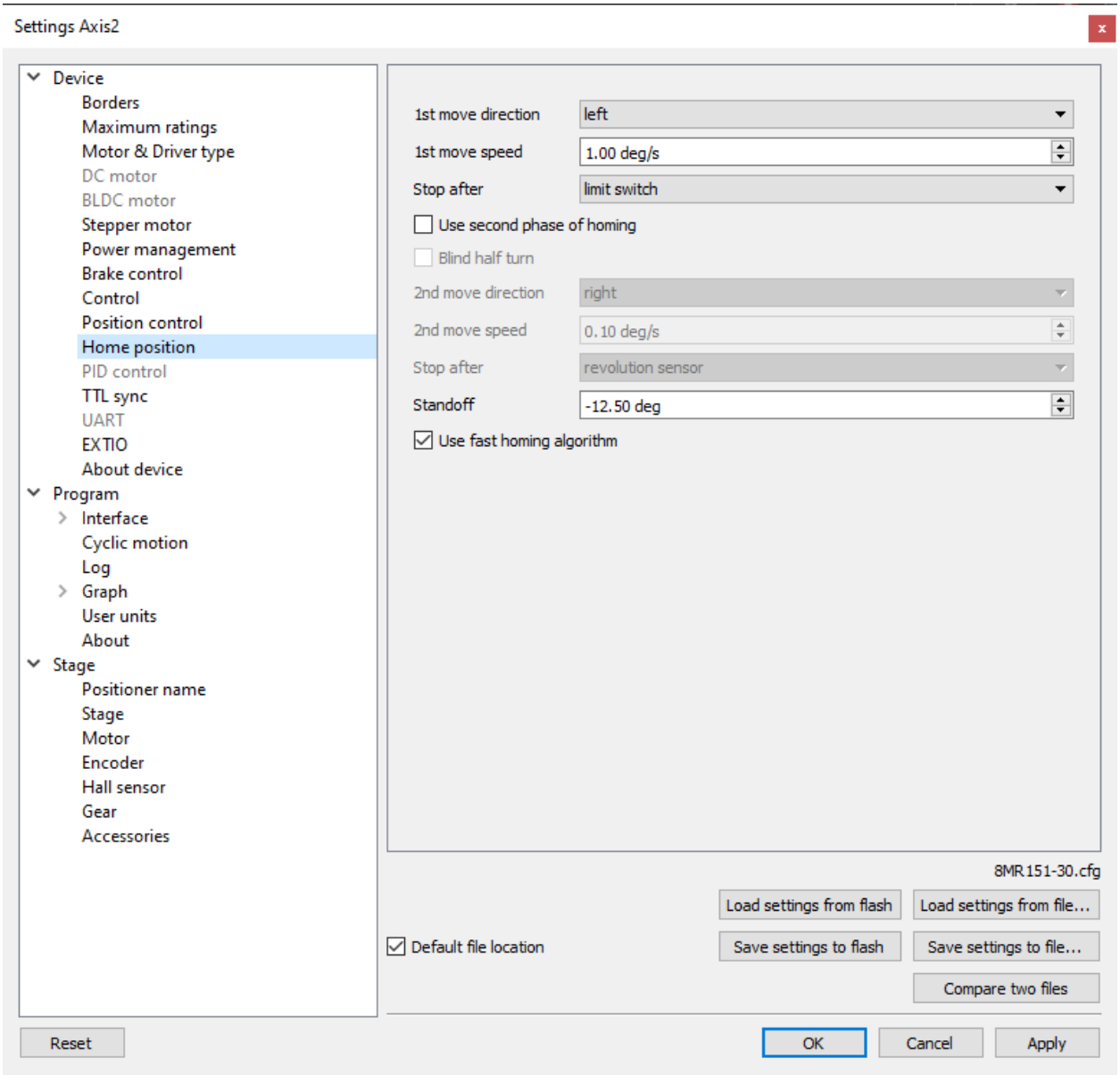


Fig. 5.25: Home position settings window

Tab **Home position** sets the *home position calibration* parameters.

- *1st move direction* - sets the movement direction for the motor to search for a stop signal (right or left) for *standard* and *fast homing algorithms*.

- *1st move speed* - sets the speed for the first phase of the *standard homing algorithm* and the second phase of the *fast homing algorithms*.
- *Stop after* - sets the source of the stop signal (*limit switch*, *revolution sensor* or external *synchronization pulse*).
- *Use second phase of homing* - this flag enables the *accurate additional calibration of the home position* (second phase of the standard calibration algorithm).
- *Blind half turn* - when the flag is set the motor ignores the stop signal for the second phase of homing for half a turn of the motor. This option allows to specify an unambiguous sequence of receiving stop signals for the first and second phases of homing in the case when the sensors are located close enough to each other.
- *2nd move direction* - sets the movement direction for the motor to search for a stop signal (right or left) for the *second phase* of the standard homing algorithm.
- *2nd move speed* - sets the speed for the second phase of the *standard homing algorithm*.
- *Stop after* (in the block of settings for the *second phase of homing*) - sets the source of the stop signal (limit switch, revolution sensor or external synchronization pulse). The signal source may differ from that used for the first phase of homing.
- *Standoff* - sets the distance for the final offset from the reference point. The direction of the offset is given by the sign of the value of this distance (a positive standoff means an offset to the right, a negative standoff - to the left).
- *Use fast homing algorithm* - this flag enables the *fast homing algorithms* to speed up the home calibration procedure.

5.3.6 Synchronization settings

In the *Application settings* **Device -> TTL sync**

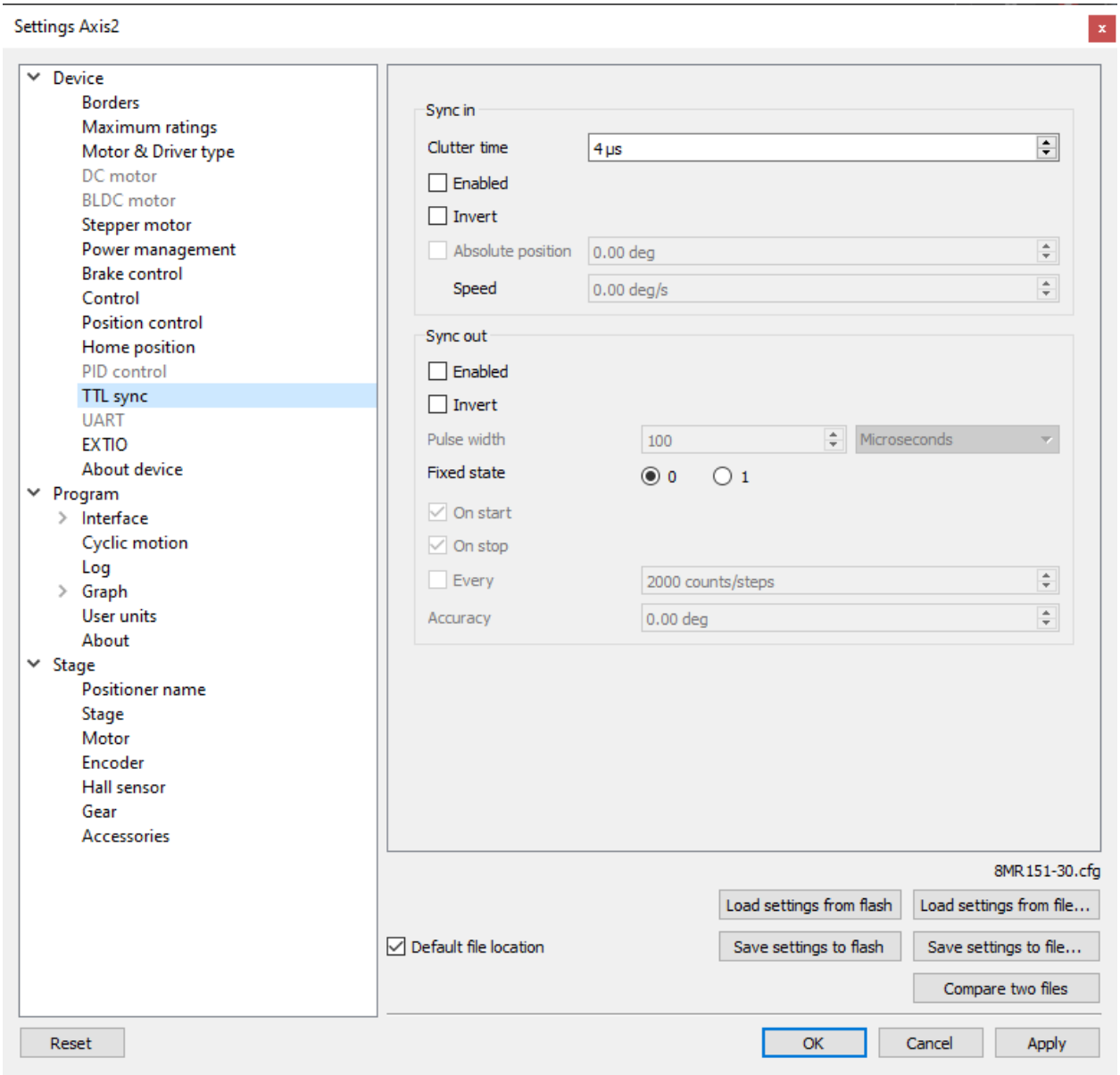


Fig. 5.26: Synchronization settings window

Synchronization is described in details in *TTL synchronization* section.

Sync in:

- *Clutter time* - setting minimum synchronization pulse duration (in microseconds). Defines the minimum duration, which can be detected (anti-chatter).
- *Enabled* - check this box for the sync in mode enable.
- *Invert* - checked flag shows that the operation is triggered by the falling sync pulse edge.
- *Absolute position* - if the flag is checked, upon sync pulse the stage moves into the absolute position specified in the field Step/Micro step. If the flag is unchecked, the shift is relative to the defined destination position.
- *Speed* - the speed to use when moving.

Important: Using the sync input pulse synchronization, to instantly start moving, you need to disable the jerk free flag, and it is also recommended to disable the power off when stop flag.

Sync out:

- The Synchronization Output can be used as a "General purpose Output signal".
- *Enabled* - if the flag is checked, the sync output functions according to the next settings. If the flag is unchecked, the output value is fixed and equal to the *Fixed state*.
- *Invert* - if the flag is checked the zero logic level is set to active.
- *Pulse width* - specifies the duration of the output signal in microseconds or steps/encoder pulses.
- *Fixed state* - sets the logic level of output to 0 or 1, respectively.
- *On start* - synchronizing pulse is generated at the beginning of movement.
- *On stop* - synchronizing pulse is generated at the end of movement.
- *Every* - synchronizing pulse is generated every n encoder pulses.
- *Accuracy* - distance to the target position. As soon as the distance to target on approach is less than or equal to this distance a synchronizing pulse will be generated if "on stop" option is used.

5.3.7 Brake settings

In the *Application settings* **Device -> Brake control**

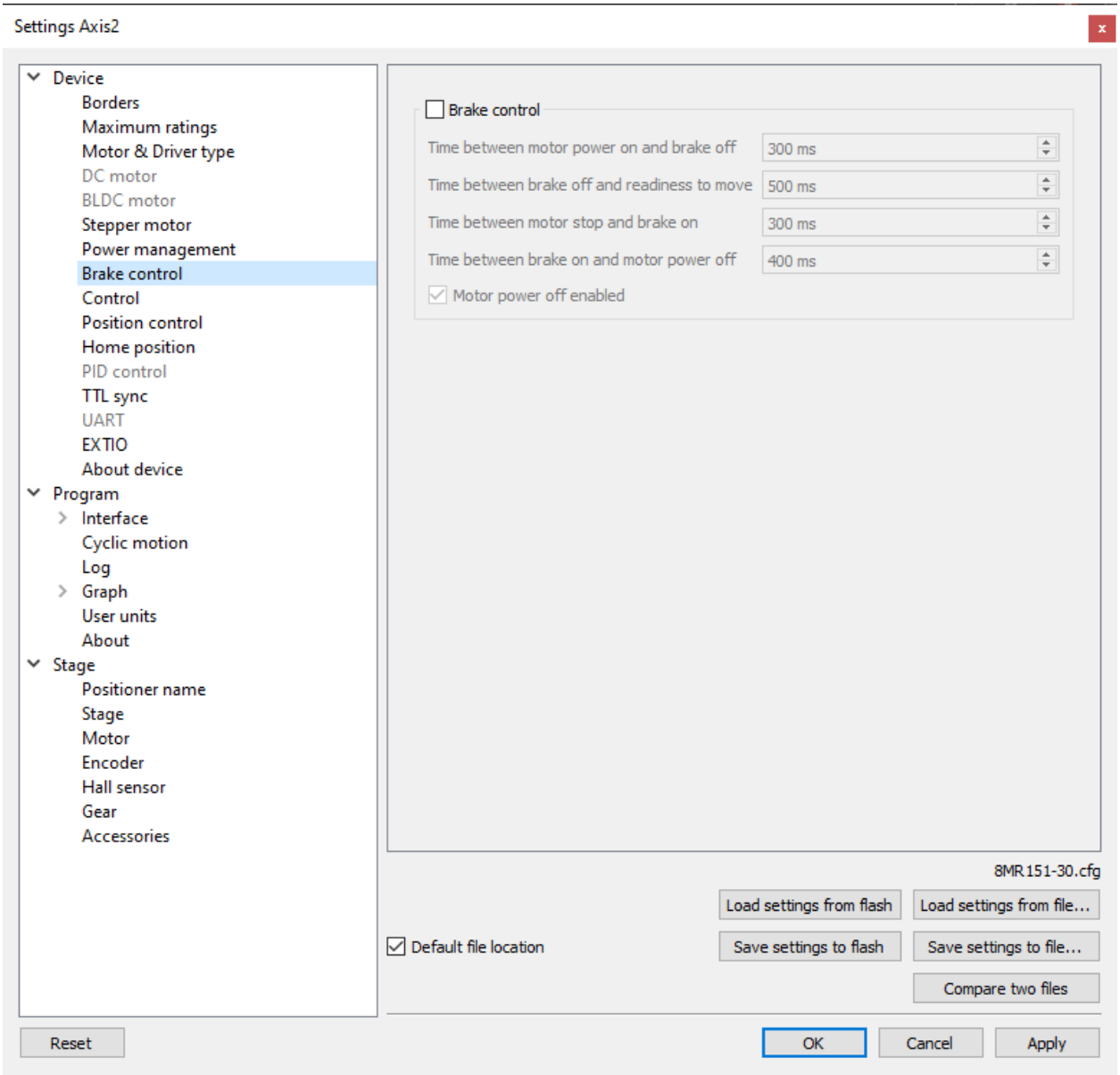


Fig. 5.27: Magnetic brake settings window

Check the *Brake control* flag to enable magnetic brake.

Parameters:

- *Time between motor power on and brake off* - time between switching the motor on and switching off the brake (ms).
- *Time between brake off and readiness to move* - time between switching off the brake and motion readiness (ms). All motion commands will be executed only after this time.
- *Time between motor stop and brake on* - time between stopping the motor and turning on the brake(ms).
- *Time between brake on and motor power off* - time between turning on the brake and motor power-off (ms). Value range is from 0 to 65535 ms.

- *Motor power off enabled* flag means that when magnetic brake is powered off, the brake turns off motor power supply.

Configuration commands are described in *Communication protocol specification*.

5.3.8 Position control

In the *Application settings* **Device -> Position control**

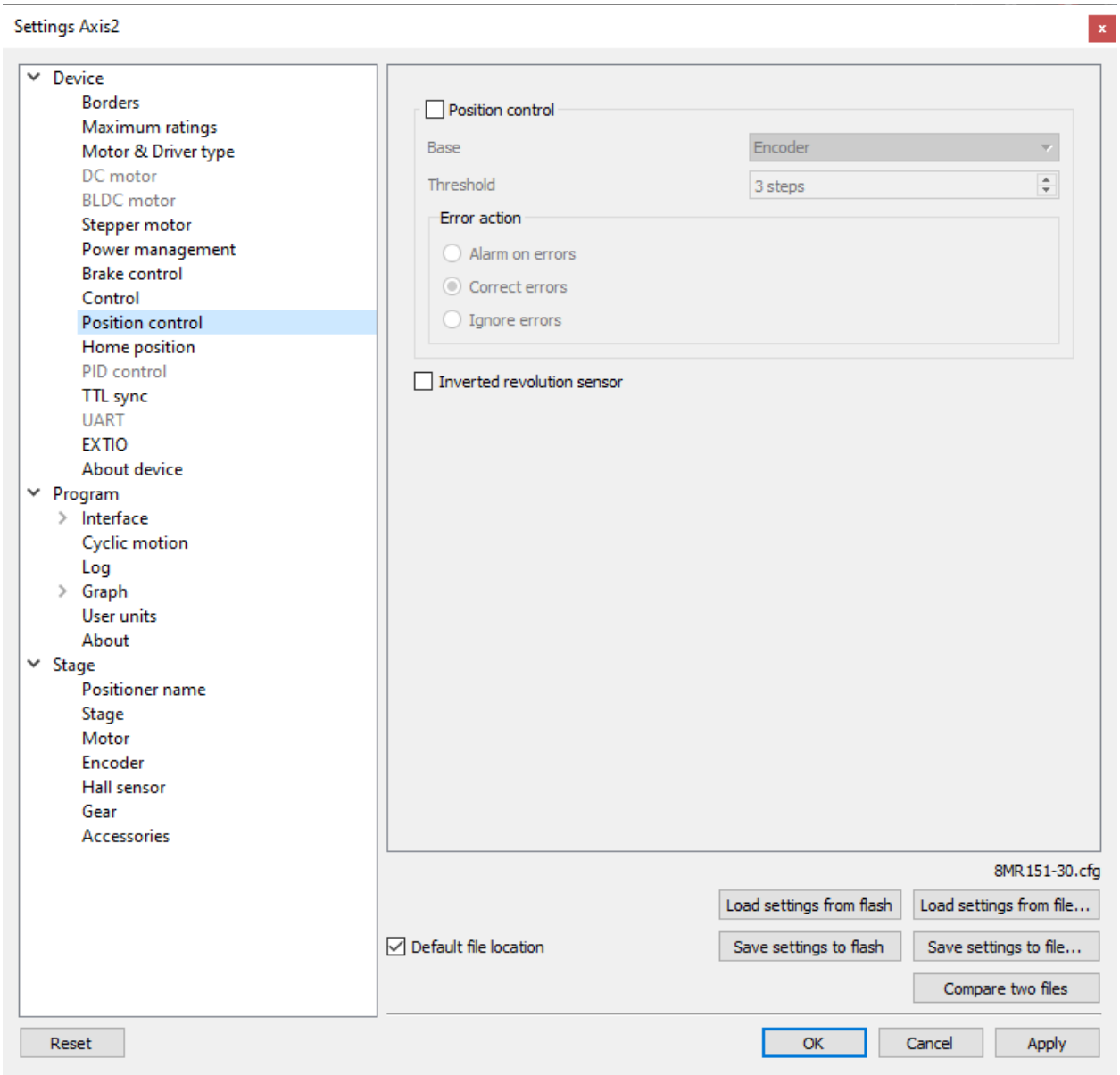


Fig. 5.28: Window of Position control

Check the *Position control* checkbox to activate the position control.

Base - selection of the position control device. You can select an encoder (see *Operation with encoders*) or revolution sensor in the drop-down list.

Minimal error - determines the number of missed steps (0 .. 255), which is considered to be an error. If the amount of missed steps exceeds the specified number of steps the SLIP error flag is set. Further actions depend on the *Error action* setting:

If *Alarm on errors* is active then the controller will enter *Alarm state*.

If *Correct errors* is active then the controller will try to correct slip error (see *Steps loss detection*).

If *Ignore errors* is active then the controller will do nothing.

Inverted revolution sensor - if the flag is checked the revolution sensor is triggered by the level 1. Unchecked flag means usual logic is valid - 0 is the trigger/activation/active state.

Configuration commands are described in the *Communication protocol specification*.

Important:

- **Feedback none**: in this mode, "Position control" is useful and should be used. "Position control" compares the position by the encoder/position sensor and recalculates it in steps. If there is a discrepancy between positions, a moving indicator "SLIP" will light up in the bottom of XiLab main window. In addition, if "Alarm on errors" is marked, the controller will enter an alarm state.
- **Feedback encoder**: "Position control" does not need to be used because the position is strictly controlled by the encoder.
- **Feedback EMF**: the algorithm should not be used with the "Position Control" flag enabled. For smooth running in the EMF algorithm, a discrepancy between the actual position and the profile position is implemented. If "Position Control" flag is enabled, false Alarms may be triggered.
- **Feedback encoder mediated**: it is not recommended to enable the "Position Control" flag. While driving, the algorithm does not differ from the "none" mode, but when the engine arrives at a position, the actual position is compared to the desired position on the encoder, after which the algorithm compensates for the discrepancy in positions until the position on the encoder becomes desirable. Thus, if the "Position Control" flag is enabled, the SLIP flag and Alarms can be triggered.

5.3.9 Settings of external control devices

In the *Application settings* **Device -> Control**

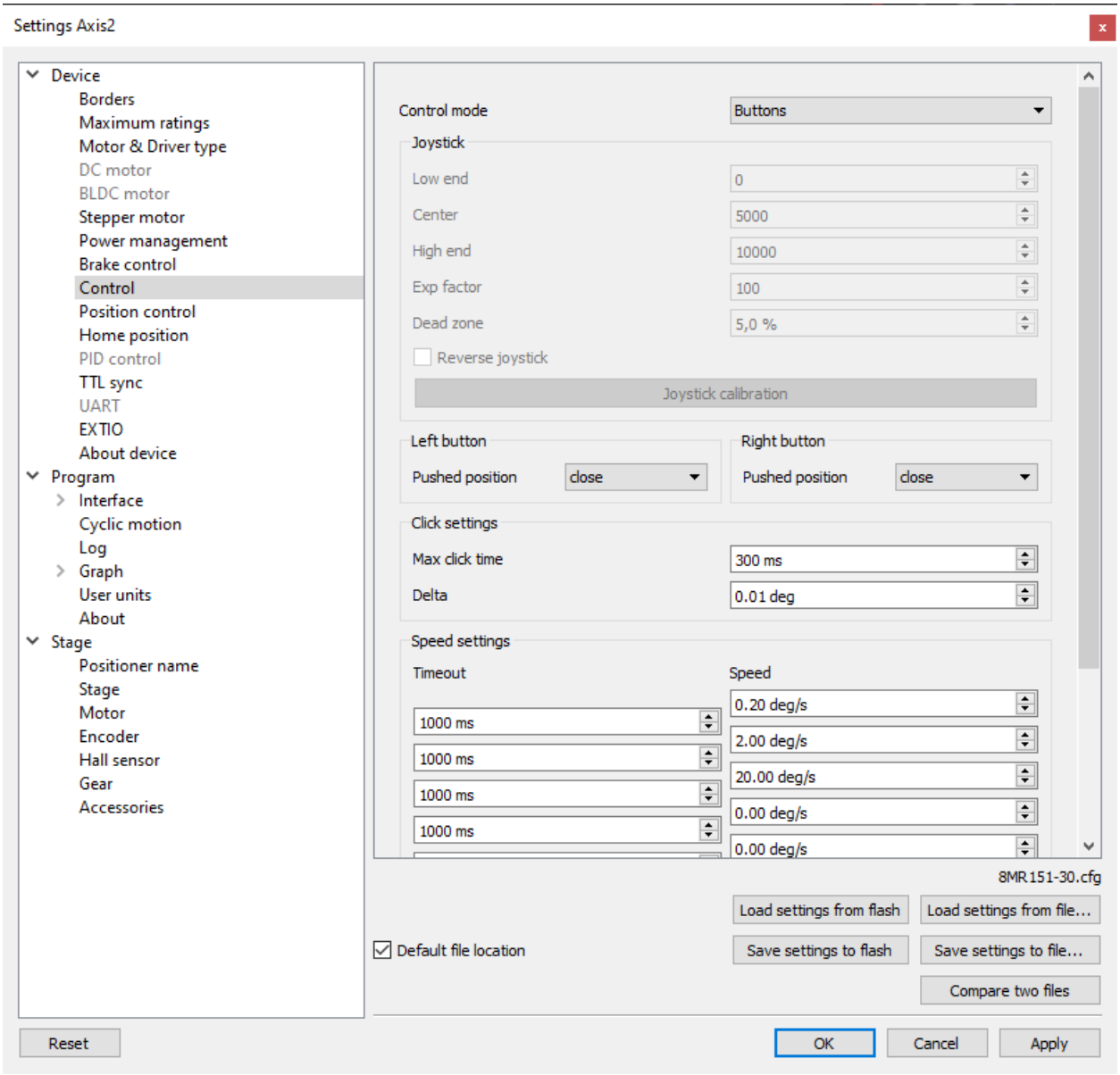


Fig. 5.29: Settings of external control devices window

Control mode - range of external motor control devices.

- *Control disabled* - external devices are not used
- *Joystick* - *joystick* is used
- *Buttons* - *buttons* are used

Important: In Joystick control mode, the physical and virtual buttons remain in working order

Joystick block contains joystick settings.

Low end, *Center* and *High end* determine the lower border, the middle and the upper border of joystick range respectively. Hence the joystick ADC normalized value equal to or less than *Low end* corresponds to the maximum joystick

deflection towards lower values.

Exp factor - exponential nonlinearity parameter. See *Joystick control*.

Dead zone - dead zone of joystick deviation from the center position. Minimum step of variation: 0.1, the maximum value is 25.5. The joystick deviation from *Center* position by less than *Dead zone* value corresponds to zero speed.

Reverse joystick - reverse the joystick effects. Joystick deviation to large values results in negative speed and vice versa.

Button *Joystick calibration* opens calibration dialog box.

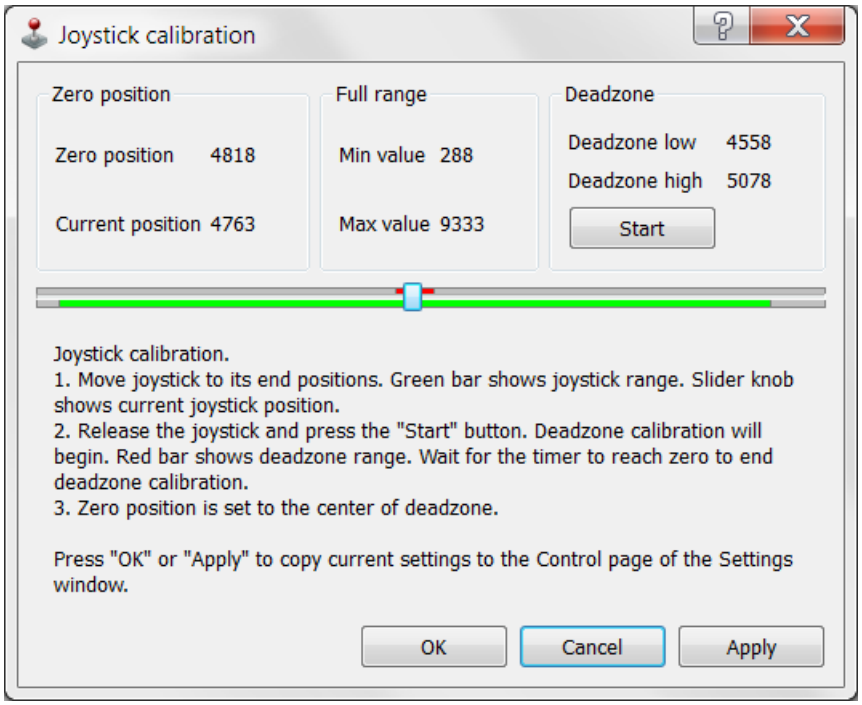


Fig. 5.30: Dialog box of Joystick Calibration

Calibration is automatic border and the dead zone detection. Below is the process description:

Move the joystick to extremes to determine the borders. The range of all measured values is represented in a green line.

Release the joystick and press the Start to initiate detection of the dead zone. Within 5 seconds imitate accidental influences on the joystick, which should not be recognized as deviation from the joystick zero position. The dead zone range is represented in red.

Pressing the Apply button will send the computed values into the Settings window. Pressing OK button will send the values and close the calibration dialog box.

Left button and **Right button** blocks contain the settings of the physical buttons.

Pushed Position - determines the state (pressed or released button) which is considered the motion signal by the controller.

- *Open* - released button is considered to be a motion command.
- *Close* - depressed button is considered to be a motion command.

Click settings block lets one to set up button “click” behaviour. A rapid press of a button is interpreted as a “click”.

Max click time - maximum click time. Until this amount of time is elapsed controller will not start moving with first speed (see below).

Delta - relative position offset. Controller will do a shift on offset with each click.

Speed settings block contains timeout and speed settings.

Timeout [i] - the time after which the speed switches from Speed[i] to Speed[i+1]. If any of the Timeout[i] is equal to zero, no switching to the next speeds will occur.

Speed[i] - speed of the motor after time equal to Timeout[i-1]. If any of the speeds is equal to zero, no switching to this and subsequent speeds will occur.

Configuration commands are described in *Communication protocol specification*.

5.3.10 General purpose input-output settings

In the *Application settings* **Device -> EXTIO**

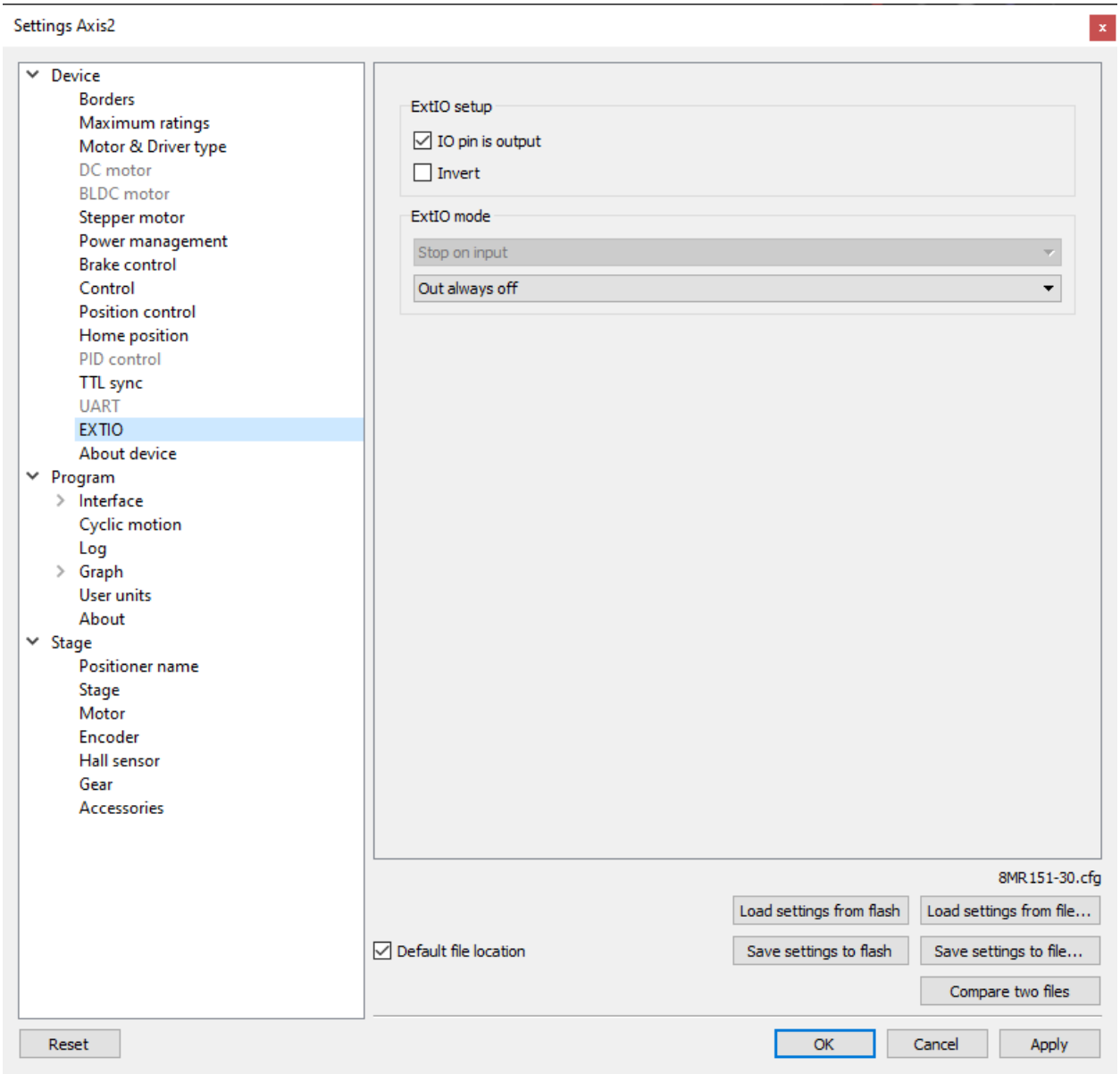


Fig. 5.31: General purpose input-output settings tab

For detailed information, please see *General purpose digital input-output (EXTIO)*.

ExtIO setup

IO pin is output - if the flag is checked the needle of ExtIO works in output mode, otherwise - in the input mode.

Invert - if the flag is checked the rising edge is ignored and the falling edge is active.

ExtIO mode - mode selection

If ExtIO configured for input mode the choice of controller action settings by the input pulse is active:

- *Do nothing* - do nothing.
- *Stop on input* - run *Command STOP*.
- *Power off on input* - run *Command PWOF*.

- *Movr on input* - run *Command MOVR*.
- *Home on input* - run *Command HOME*.
- *Alarm on input* - enter ALARM state.

If ExtIO configured for output mode the choice of the output state depending on the controller status is active:

- *Out always off* - always in inactive state.
- *Out always on* - always in active state.
- *Out active when moving* - in active state during motion.
- *Out active in Alarm* - in active state if the controller is in the Alarm state.
- *Out active when motor is on* - in active state if the motor windings are powered.
- *Out active when motor is found* - in active state if the motor is connected.

5.3.11 Motor type settings

In the *Application settings* **Device -> Motor & Driver type**

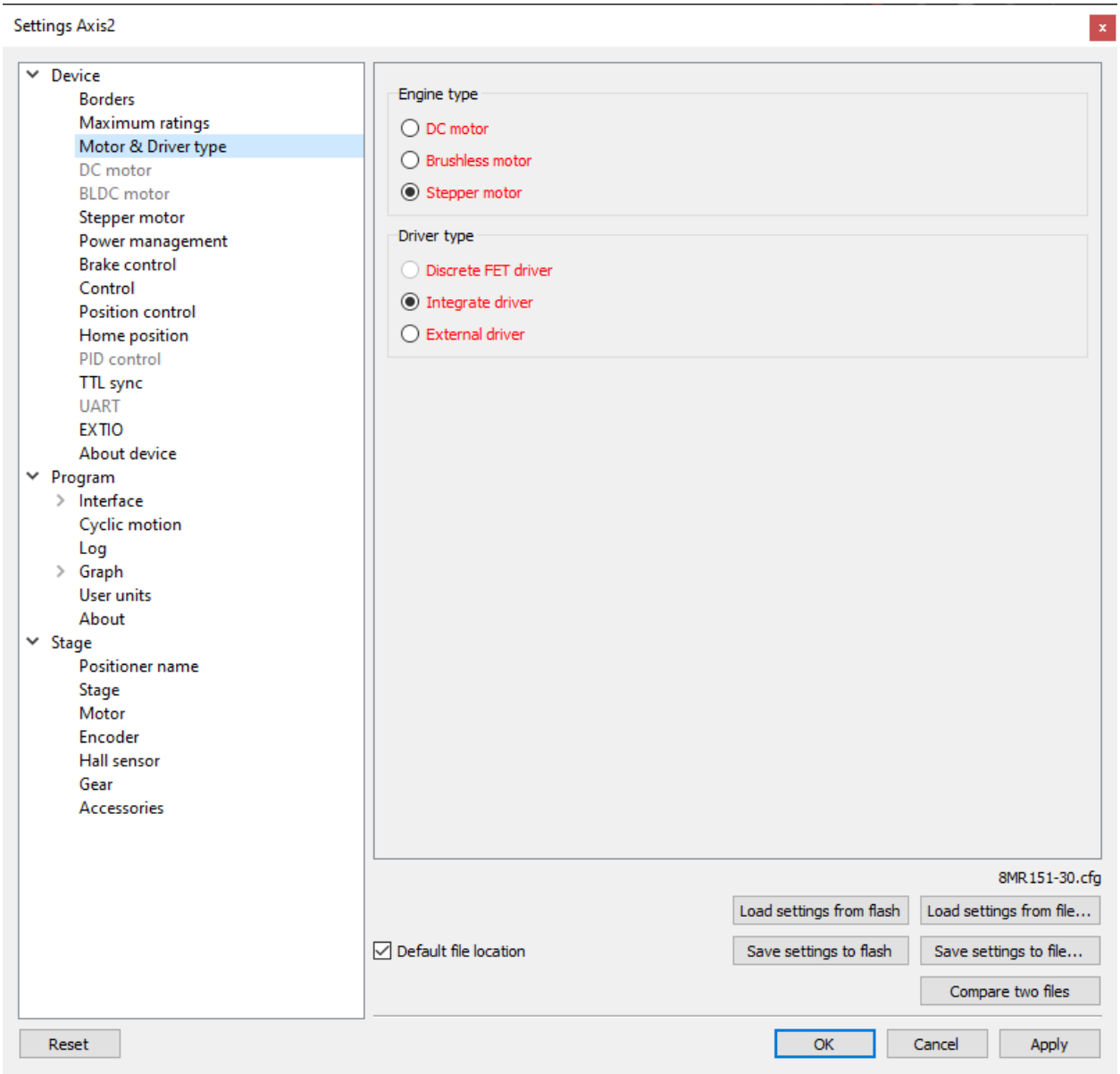


Fig. 5.32: Motor type settings window

Stepper motor or *DC-motor* - motor type indication. Control power driver should be selected as well:

- Integrated. This type is used for this controller modification.
- Discrete FET driver. Will be used in future versions.
- *External driver*. Designed to control stepper motors using three standard signals (see *External driver control interface*)

Warning: Driver type or motor type change is a critical operation that should not be performed while motor rotates. To implement the change correctly the motor winding should be de-energized and turned off, after that motor type can be changed and motor of another type can be connected. The same applies to changing of integrated driver to external one and vice versa.

Note: Available motor types are determined by your firmware upgrade. Available control drivers depend on the controller board type, except for the external driver.

5.3.12 Settings of kinematics (DC motor)

In the *Application settings* **Device -> DC Motor**

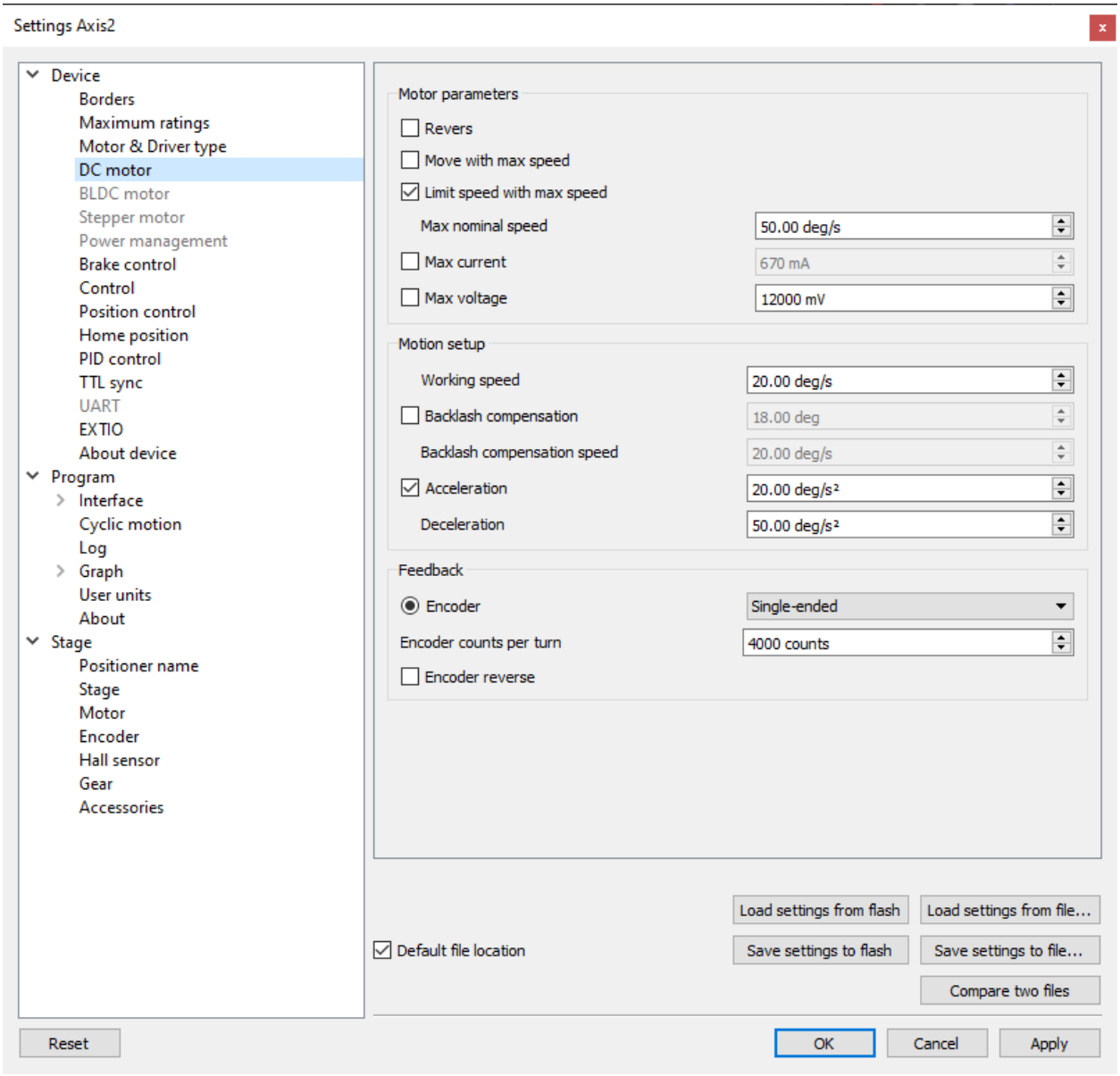


Fig. 5.33: Settings of kinematics (DC motor) window

5.3.12.1 Motor parameters - electric motor settings

Revers - checking this flag associate the motor rotation direction with the current position counting direction. Change the status of the flag if positive motor rotation decreases the value on the position counter register. This flag effect is similar to connecting the motor winding to reverse polarity.

Move with max speed - if this flag is checked motor ignores the preset speed and rotates at the maximum speed limit.

Limit speed with max speed - if this flag is checked the controller limits the maximum speed to the number of steps per second, specified in the *Max nominal speed* field.

Max nominal speed, Max voltage, Max current - are motor nominal parameters. If they are active and applicable for given type of motor, the controller limits these parameters within the specified values. For example, if the motor speed and voltage exceeds the nominal values, the controller will reduce output action until both values are within the normal range. However, the controller remains in operational condition, and will execute the current task.

5.3.12.2 Motion setup - settings related to the movement kinematics

Working speed - movement speed.

Backlash compensation - backlash compensation. Since the stage mechanics are not ideal there is a difference between approaching a given point from the right and from the left. When the backlash compensation mode is on the stage always approaches the point from one side. The preset value determines the number of steps which the stage takes to pass a given point in order to come back to it from the same side. If the specified number is above zero the stage always approaches the point from the right. If it is below zero the stage always approaches the point from the left.

Backlash compensation speed - speed of backlash compensation. When the backlash compensation mode *Backlash compensation* is on the stage approaches the point from the right or from the left with a preset speed determined in the number of steps per second.

Acceleration - enables the motion in acceleration mode, the numerical value of the field is the acceleration of movement.

Deceleration - movement deceleration.

5.3.12.3 Feedback settings

Encoder - use of *encoder* as a feedback sensor. The following encoder types are available: Single-ended, Differential or Autodetect.

Encoder counts per turn - this parameter defines the number of encoder pulses per one motor axis full rotation.

5.3.13 Settings of PID control loops

In the *Application settings* **Device -> PID control**

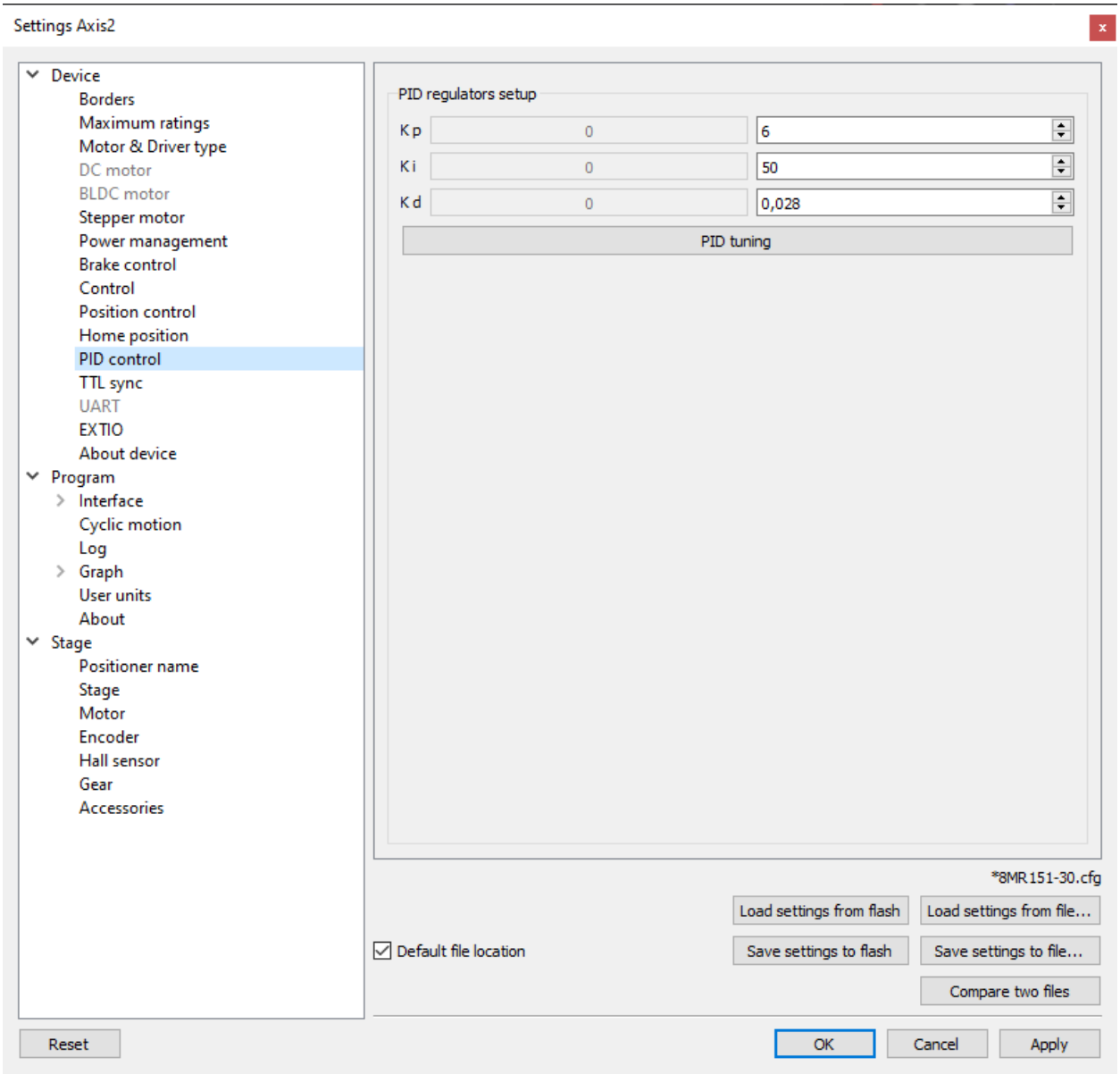


Fig. 5.34: Settings of PID control loops window

In this section, you can change the PID coefficients. A voltage PID is used, K_p, K_i and K_d coefficients can vary in 0..65535 range for DC motors.

Fractional PID coefficients (on right) are only for BLDC motor (supported in firmware 4.1.x and older).

Warning: Do not change the settings of PID controllers, if you are not sure you know what you are doing!

Configuration commands are described in the *Communication protocol specification*. PID tuning is described in detail in the *PID-algorithm for DC engine control* section.

5.3.14 About controller

In the *Application settings* **Device -> About device**

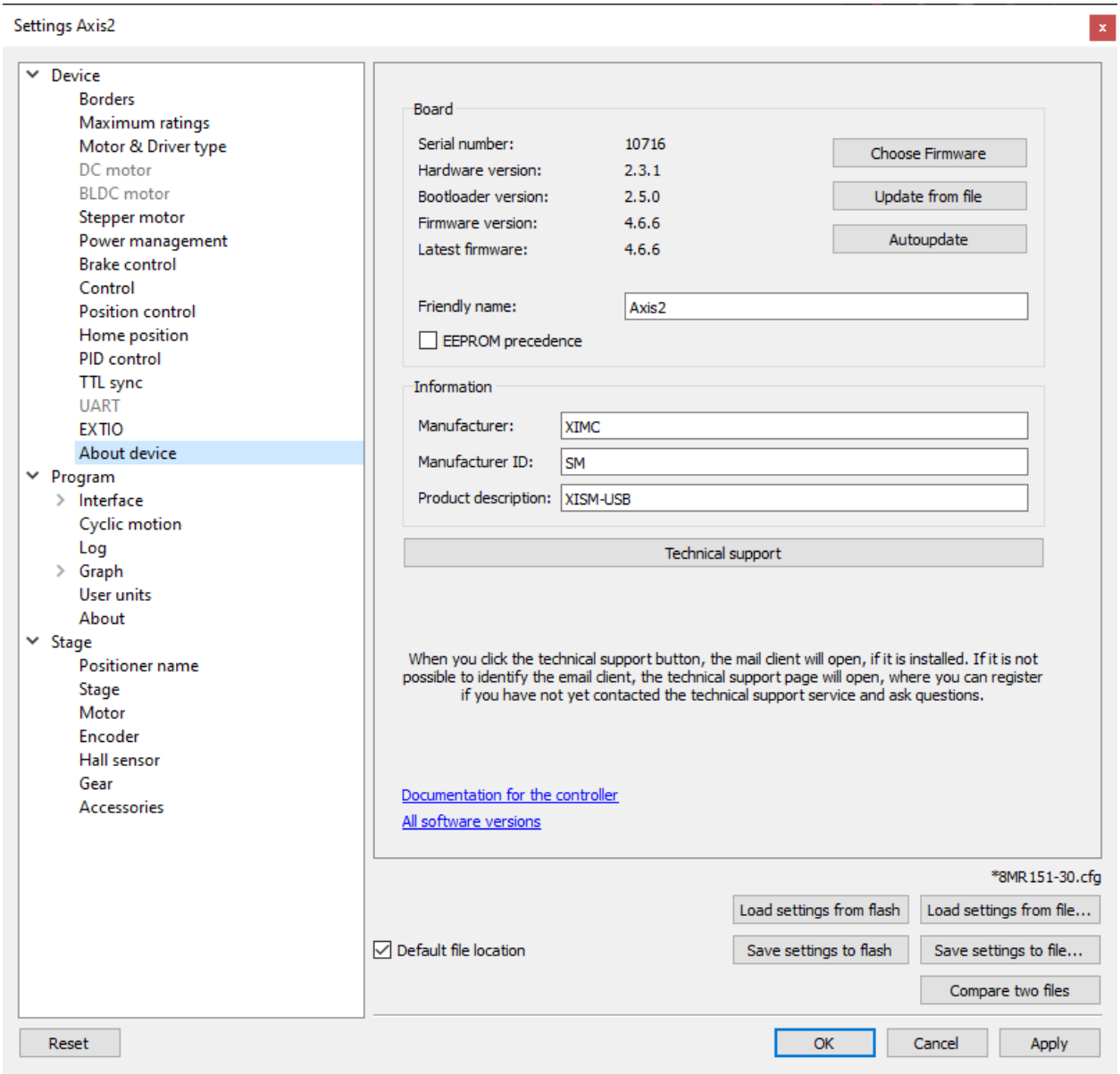


Fig. 5.35: About device tab

The **Board** section displays information about the controller:

- *Serial number* - device serial number.
- *Bootloader version* - bootloader version.
- *Firmware version* - firmware version.
- *Latest firmware* - latest available firmware version for this device (downloaded from the internet if internet connection is available).
- The *Update* button opens firmware update dialog box.

Select the firmware file with the .cod extension and click Open. XILab will start the firmware update and will display "Please wait while firmware is updating". Do not power off the controller during the upgrade. Upon completion of the update the "Firmware updated successfully" dialog will be displayed.

The *Choose Firmware* button opens a dialog with firmware version numbers. Pick a number and press "Update firmware" to update to selected version. An appropriate firmware file will be downloaded from the internet and loaded into the controller. This feature requires an active internet connection.

The *Autoupdate* button automatically updates firmware from the internet to the latest available version.

Friendly name - an arbitrary user-defined name for the controller. If this string is not empty, then it will replace device id and serial number in window titles. This is a convenience feature for situations with multiple controllers connected to the same PC.

EEPROM precedence - this flag works only for stages with *autodetection feature*. If this flag is set, then settings from external EEPROM memory take precedence and are applied every controller start up or stage connection. Otherwise settings from FRAM are used. If this option is turned on then "Save settings to flash" button is displayed in red. This serves as a warning to the user that settings in flash memory will be overwritten when a device with external memory is plugged into the controller.

Information block contains information about the device: the manufacturer, device ID, device type. The data are read from the internal memory of the controller.

All of this data is reported to the XILab application when the device is connected.

5.3.15 Settings of kinematics (BLDC motor)

In the *Application settings* **Device -> BLDC Motor**

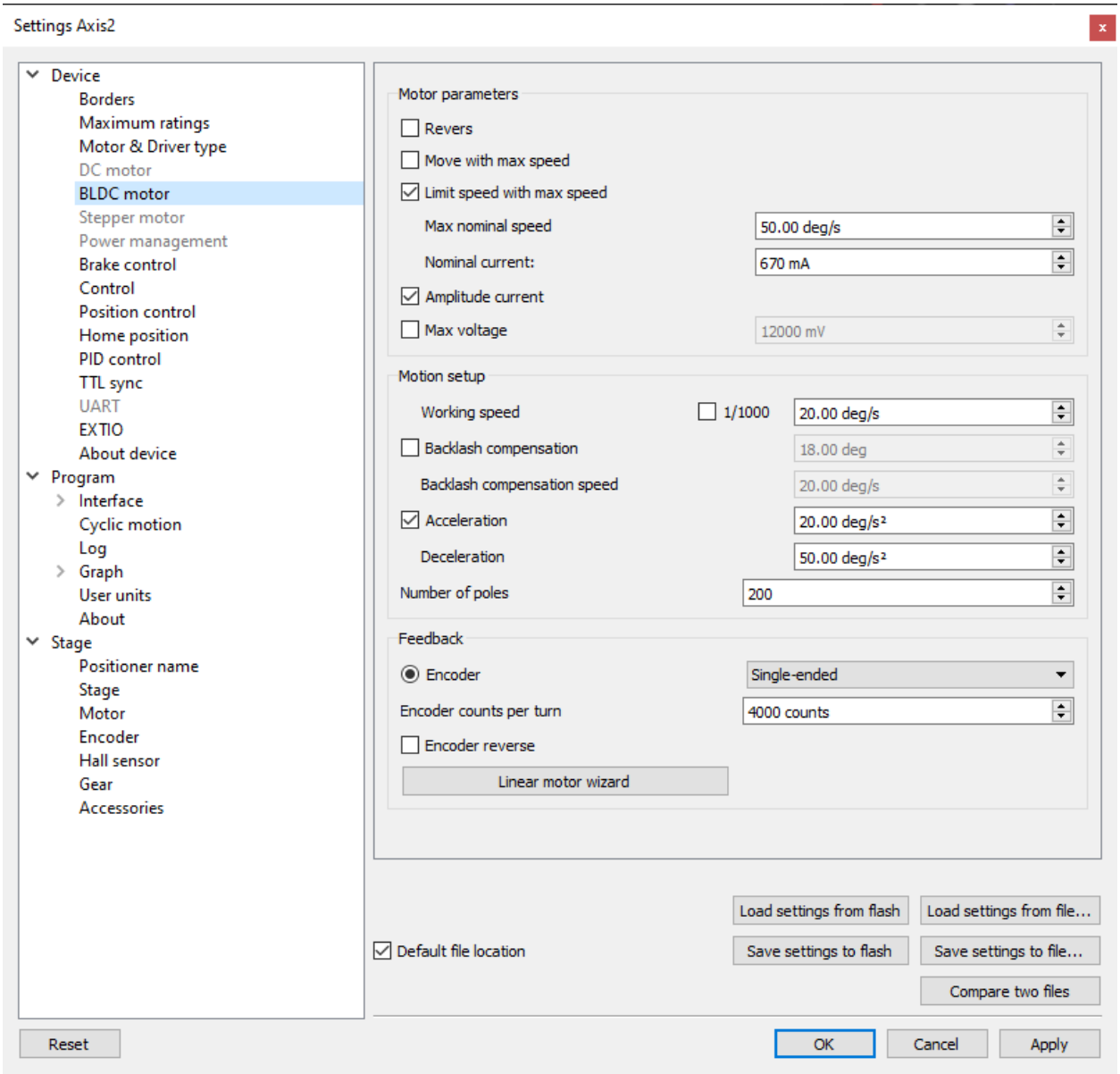


Fig. 5.36: Settings of kinematics (DC motor) window

Important: Only firmwares 4.1.x (and older) support BLDC control.

5.3.15.1 Motor parameters - electric motor settings

Revers - checking this flag associate the motor rotation direction with the current position counting direction. Change the status of the flag if positive motor rotation decreases the value on the position counter register. This flag effect is similar to connecting the motor winding to reverse polarity.

Move with max speed - if this flag is checked motor ignores the preset speed and rotates at the maximum speed limit.

Limit speed with max speed - if this flag is checked the controller limits the maximum speed to the number of steps per second, specified in the *Max nominal speed* field.

Max nominal speed, Max voltage - are motor nominal parameters. If they are active and applicable for given type of motor, the controller limits these parameters within the specified values. For example, if the motor speed and current exceeds the nominal values, the controller will reduce output action until both values are within the normal range. However, the controller remains in operational condition, and will execute the current task.

Important: "Max voltage" is the maximum voltage between any two terminals of the BLDC motor. At the same time, the voltage at each terminal relative to the controller's ground (this voltage is shown on the graphs "Winding A Voltage", "Winding B Voltage") may exceed "Max voltage".

Amplitude current - if this flag is checked engine current value is interpreted as maximum amplitude value. If the flag is unset, then engine current value is interpreted as the current value calculated from the maximum heat dissipation. See *Calculation of the nominal current* page for description

5.3.15.2 Motion setup - settings related to the movement kinematics

Working speed - movement speed.

Flag 1/1000 - allows you to work at ultra-low speeds. If the flag is enabled, the value from the "Working speed" field will be divided by 1000.

Important: The flag for working with ultra-low speeds was added in xilab 1.17.9 (and older) and firmware 4.5.9 (and older)

Backlash compensation - backlash compensation. Since the stage mechanics are not ideal there is a difference between approaching a given point from the right and from the left. When the backlash compensation mode is on the stage always approaches the point from one side. The preset value determines the number of steps which the stage takes to pass a given point in order to come back to it from the same side. If the specified number is above zero the stage always approaches the point from the right. If it is below zero the stage always approaches the point from the left.

Backlash compensation speed - speed of backlash compensation. When the backlash compensation mode *Backlash compensation* is on the stage approaches the point from the right or from the left with a preset speed determined in the number of steps per second.

Acceleration - enables the motion in acceleration mode, the numerical value of the field is the acceleration of movement.

Deceleration - movement deceleration.

Number of poles - number of poles per revolution.

5.3.15.3 Feedback settings

Encoder - use of *encoder* as a feedback sensor. The following encoder types are available: Single-ended, Differential or Autodetect.

Encoder counts per turn - this parameter defines the number of encoder pulses per one motor axis full rotation.

Linear motor wizard - open dialog for setting linear stages parameters.

5.4 XILab application settings

5.4.1 XILab general settings

Program in the *Application settings*

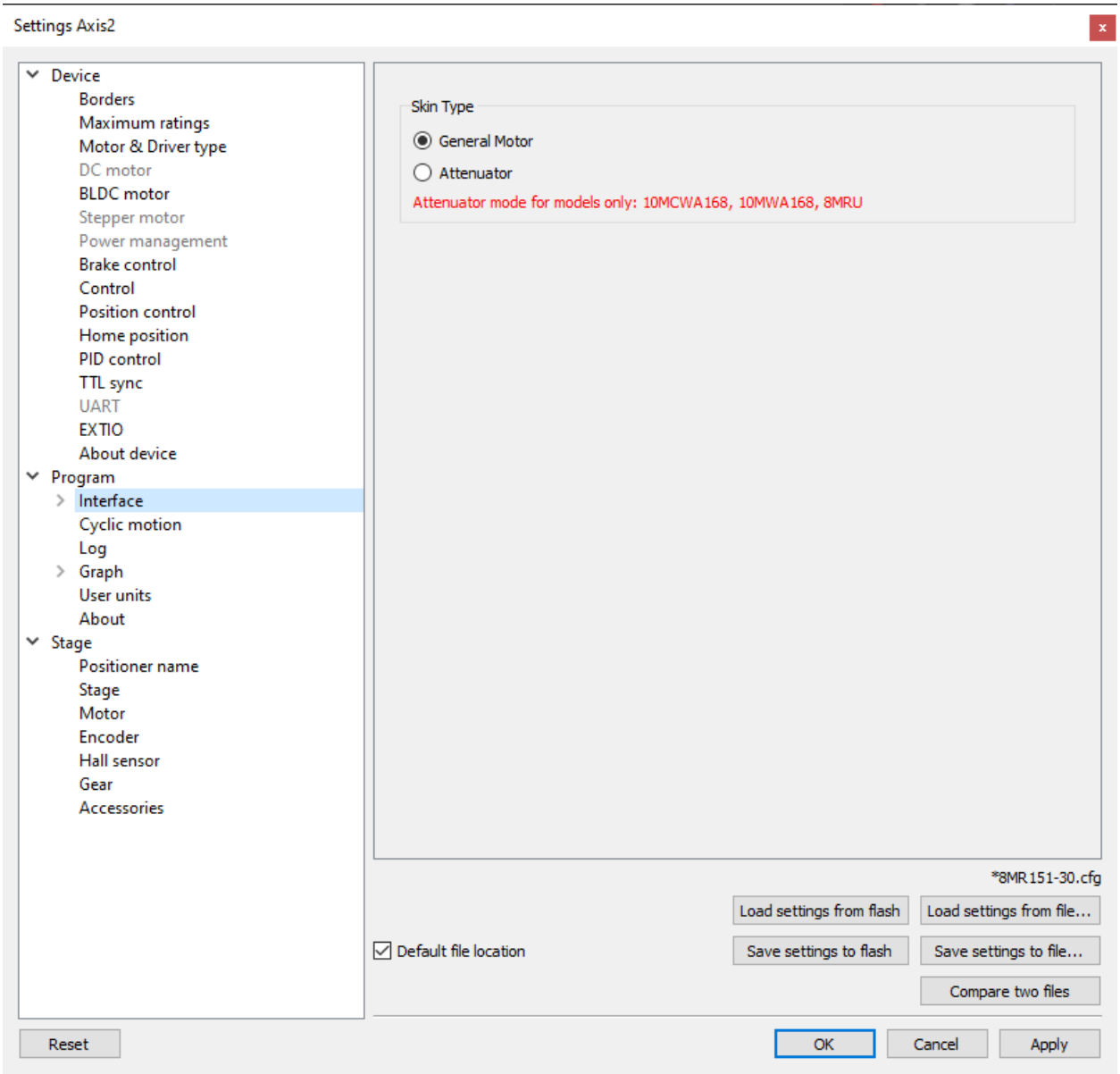


Fig. 5.37: XILab general settings tab

This tab configures the Xilab interface type.

Skin Type group includes Xilab interface type settings. There are two interface types available: “General Motor” and “Attenuator”.

General Motor option enables general motor interface. In this mode current motor position is displayed in main window numerically and graphically as a slider. Controls to move the motor to specified coordinate (in steps) and to shift on specified offset (in steps) are also available. General motor interface settings are located on the *General motor settings* page.

Attenuator option enables Attenuator interface. In this mode current motor position is displayed in main window graphically as a series of circles representing attenuator filters. Controls to find the combination of filters which is the best approximation to the specified transparency are also available. Attenuator interface settings are located on the *Attenuator settings* page.

5.4.2 General motor settings

In the *Application settings* **Program -> Interface -> General motor**

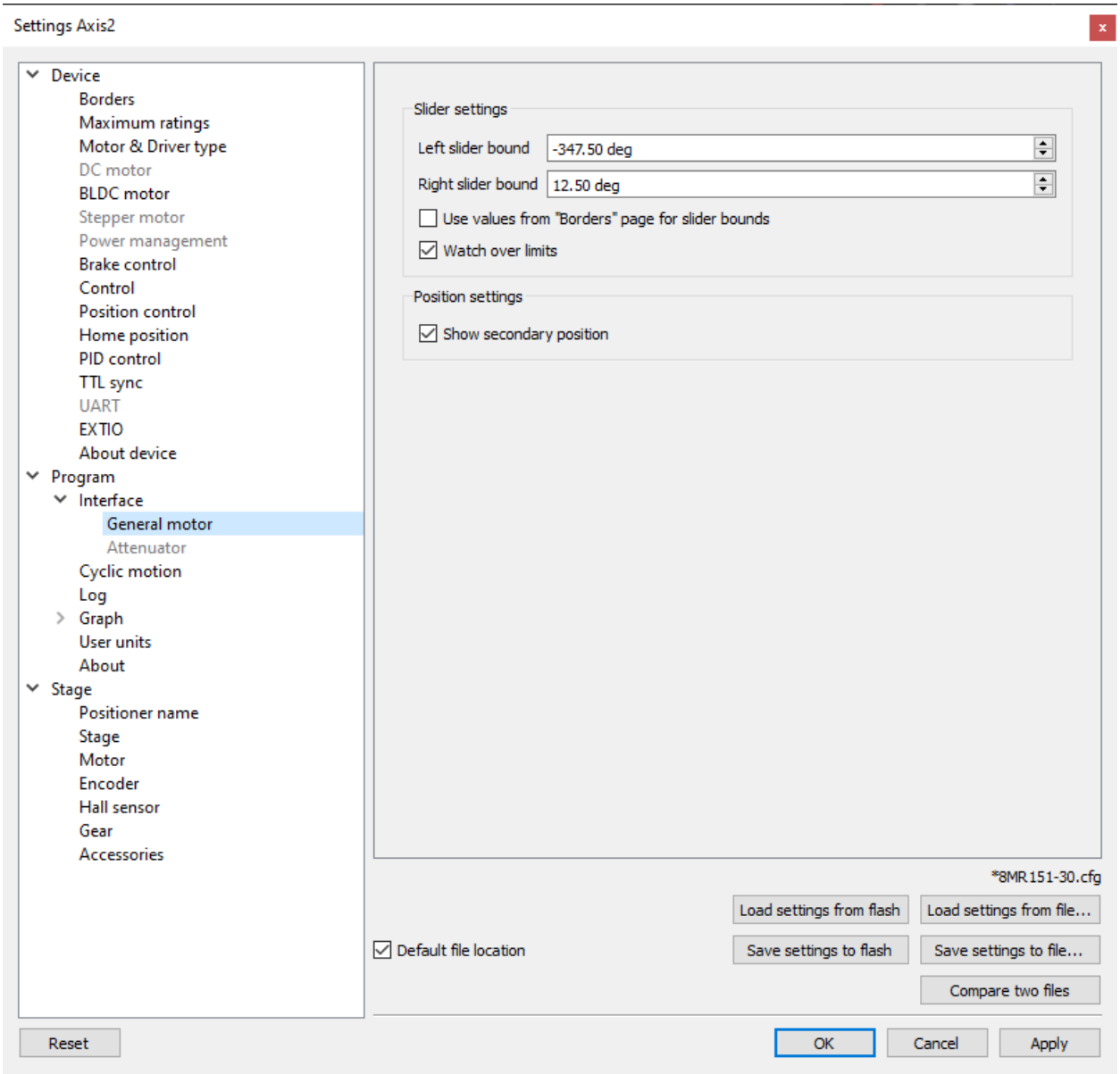


Fig. 5.38: General motor settings tab

This tab configures the slider display settings and secondary position display settings for a general motor device. The position slider is located in the *main window* and visually represents the stage current position relative to the borders.



Fig. 5.39: Fragment of Main application window containing slider

Slider settings group contains the following slider settings:

Left slider bound and *Right slider bound* contain the coordinates of the left and right bounds of the slider respectively.

If *Watch over ltimits* is checked then upon moving out of the slider range, the scale shifts to display the current position. However, he total distance displayed on the slider remains unchanged. This option is not used by default. It is useful when you know the stage motion range, but do not know the relation of that position to the values displayed in XILab, e.g. for the calibration purposes. The option is often used together with the settings of the tab *Home position settings*.

Position settings group contains the position display settings.

If *Show secondary position* is checked then a secondary position is shown in the main application window.

5.4.3 Attenuator settings

In the *Application settings* **Program -> Interface -> Attenuator**

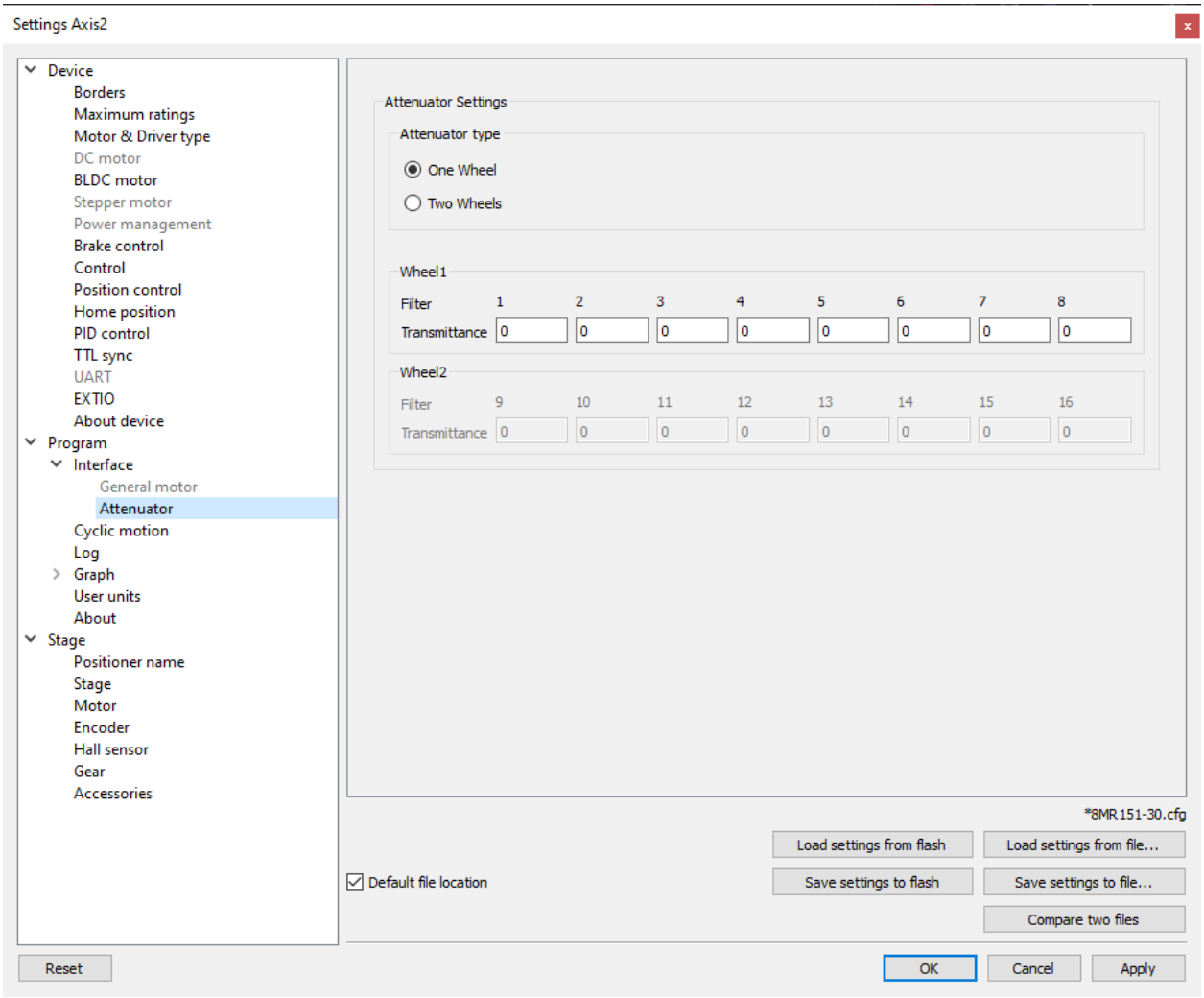


Fig. 5.40: Attenuator settings tab

This tab is used to configure attenuator interface. An attenuator is a device used to reduce the power level of an optical signal by passing light through one or several optical filters. This tab becomes active when an “Attenuator” skin type is selected in “Program” tab.

Attenuator type group contains radio buttons which allow one to choose one- or two-wheel configuration of the attenuator. Depending on the choice here one or two **Wheel** groups become available. They should be filled with attenuator filters’ transparency values on the corresponding wheel, in order.

5.4.4 Cyclical motion settings

In the *Application settings* **Program -> Cyclic motion**

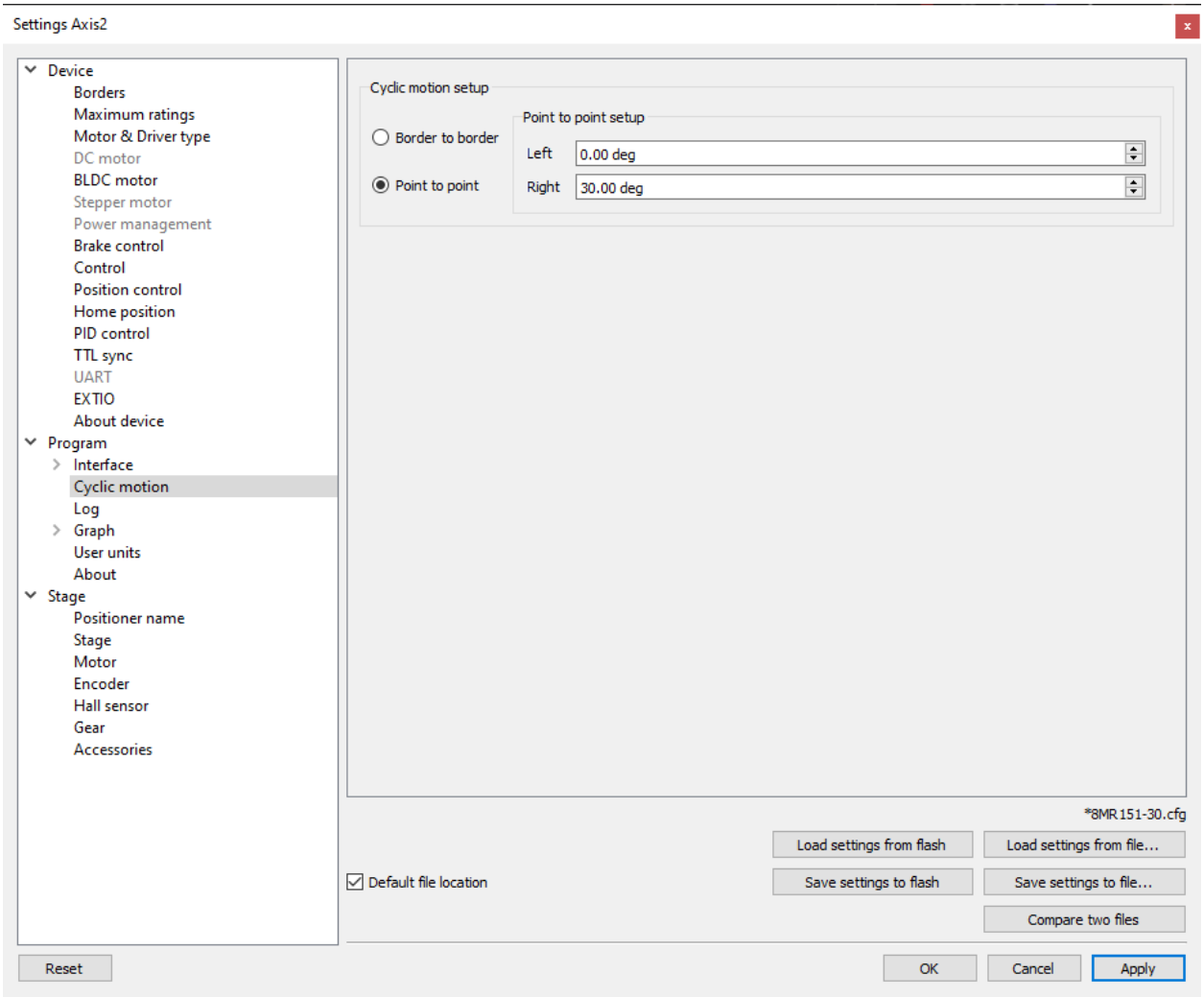


Fig. 5.41: Cyclic motion tab

Use this tab to configure the cyclic motion between two preset positions. It is used mainly for demonstration purposes. This mode is activated by *Cyclic* button in the *main window*, and deactivated by *Stop* button in the *main window*.

Cyclic motion mode settings:

Border to border - cyclical motion between the borders configured in the *Motion range and limit switches*. The motion begins towards the left edge.

Point to point - cyclical motion between points specified in the *Point to point setup* group. The stage moves to the left point, stops, then moves to the right point, stops, and then the cycle repeats.

5.4.5 Log settings

In the *Application settings* **Program -> Log**

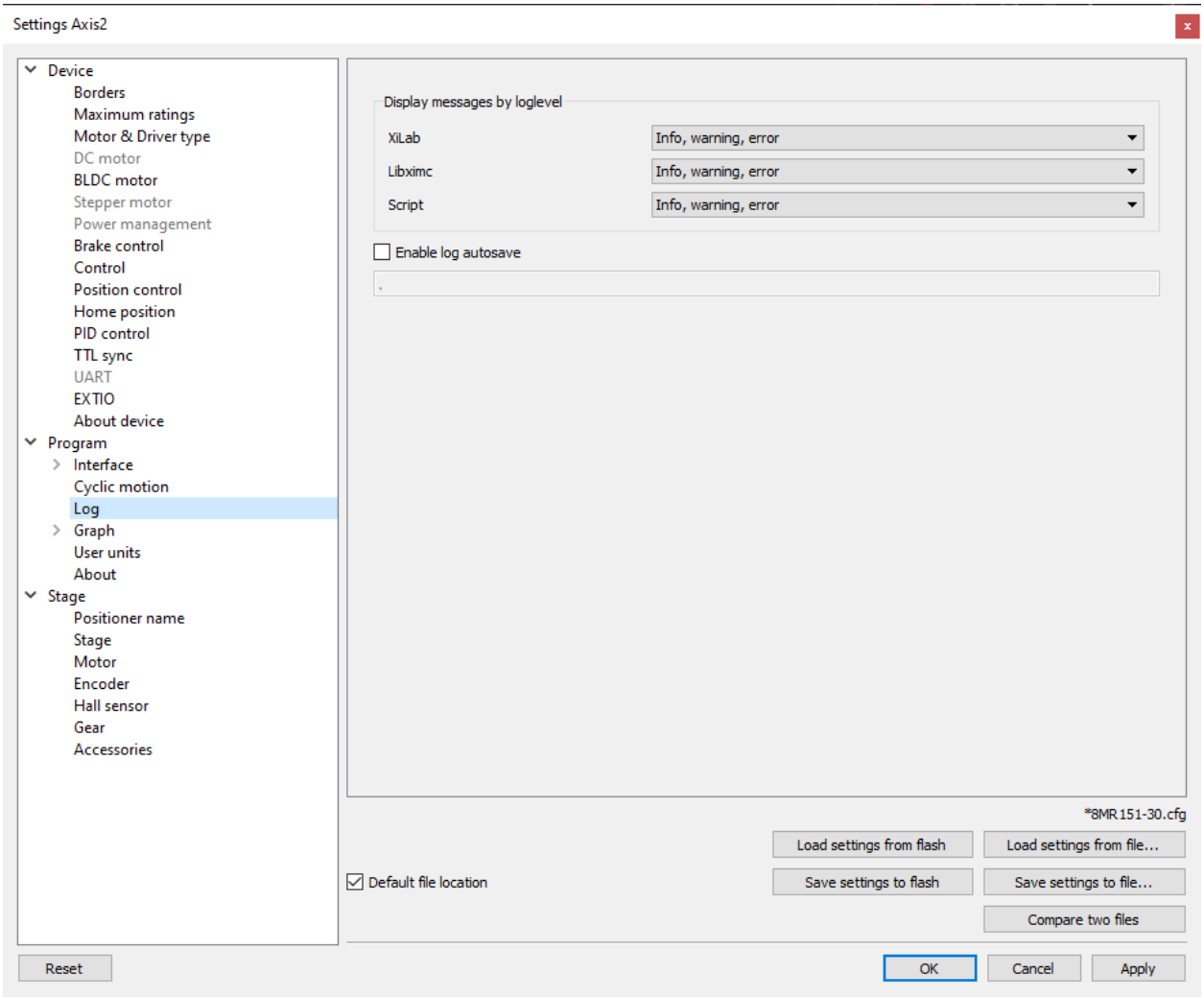


Fig. 5.42: XILab log settings window

On this tab you can configure the logging detail level.

In *Display messages by loglevel* box you can choose an option to log nothing (None), log only errors (Error), errors and warning messages (Error, Warning), errors, warnings and information messages (Error, Warning, Info) for each source: XILab application, libximc library and Scripts module.

If the *Enable log autosave* checkbox is checked then the log is saved into file. Directory where the log file will be saved is set below. Log file is flushed to the disk every 5 seconds.

File has a name of type "xilab_log_YYYY.MM.DD.csv", where YYYY, MM and DD are current year, month and day, respectively. Data is stored in CSV format. Messages that are saved into the log file are not filtered by logging options.

5.4.6 Charts general settings

Program -> Graph in the *Application settings*

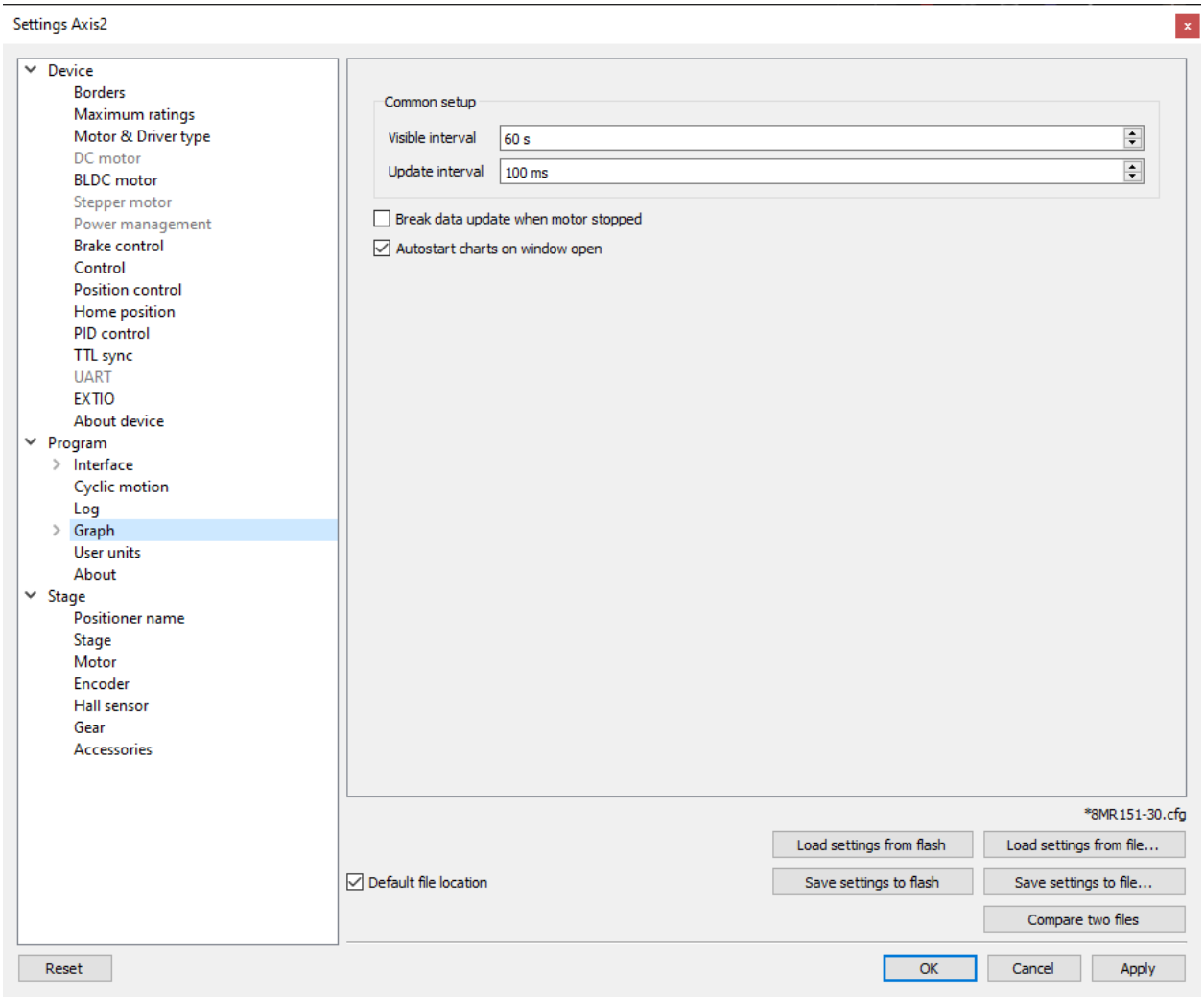


Fig. 5.43: Charts general settings tab

Visible interval - the time interval displayed in charts on the horizontal axis.

Update interval - chart data update interval.

Break data update when motor stopped - stops drawing charts when the motor stops. This option provides the possibility to use the chart space more rationally, removing the areas when there is no motor motion.

Autostart charts on window open - starts displaying chart data automatically on window open. If you wish to start charts update manually, then uncheck this option.

5.4.7 Charts customization

In the *Application settings* **Program -> Graph -> . . .**

This section equally applies to the individual appearance settings of speed, voltage, current, PWM duty factor, temperature, and other parameter charts, which can be displayed in the XILab application.

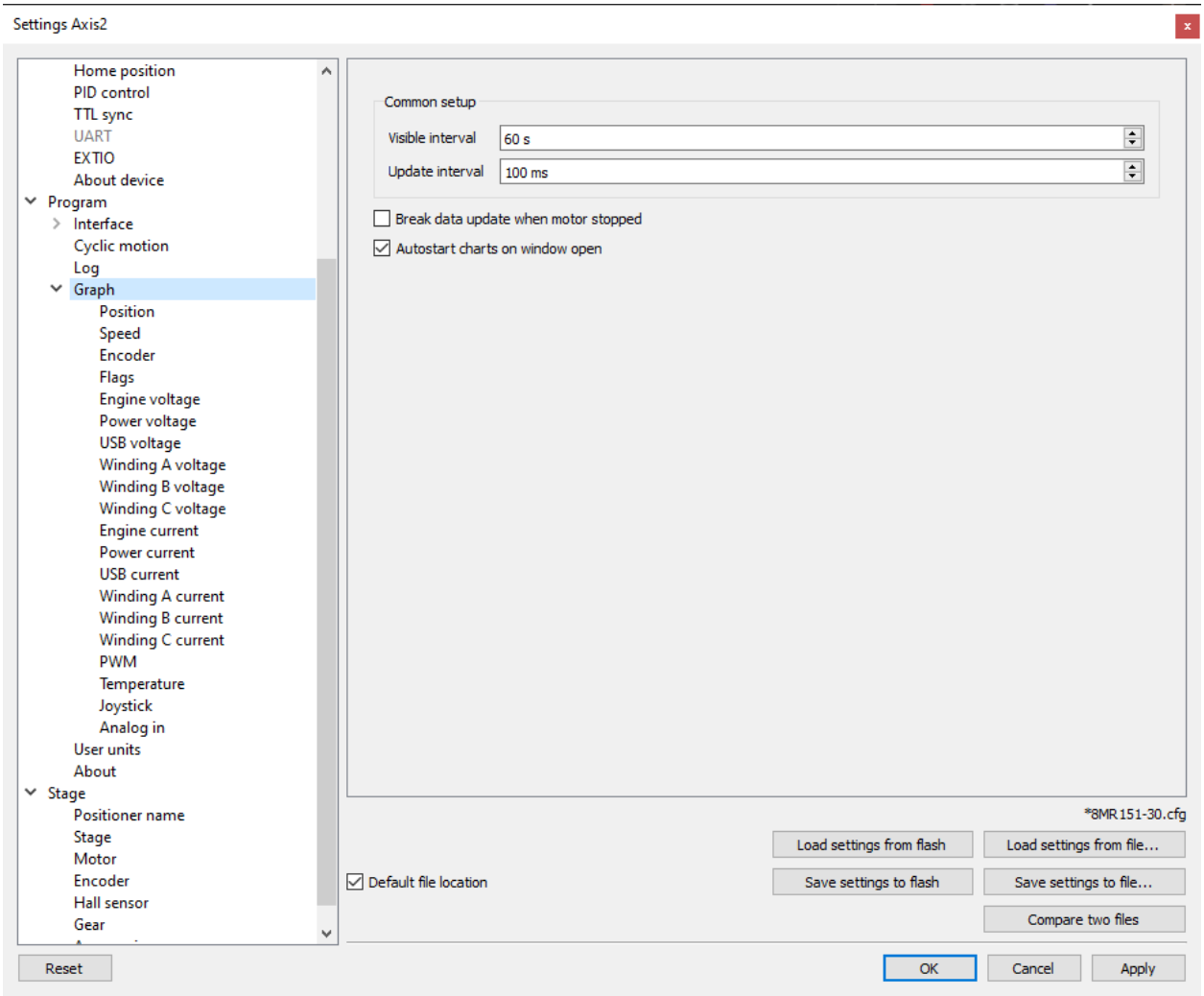


Fig. 5.44: Charts customization on the example of the position chart tab

Charts display settings include line style and chart vertical axis scale adjustment.

Position curve setup group changes curve parameters. It includes the *Line width*, *Color* and *Line style*.

Scaling group changes curve display range on the vertical axis by setting values in *Scale min* and *Scale max*.

Checked *Autoscale* flag results in auto-scaling of the scale limits in accordance with the change limits of the variable on the axis Y. In this case, the parameters *Scale min* and *Scale max* are ignored.

Antialiasing flag enables chart lines smoothing, which provides the possibility to achieve a higher-quality display, but it slows a little the chart drawing process.

5.4.8 User units settings

In the *Application settings* **Program -> User units**

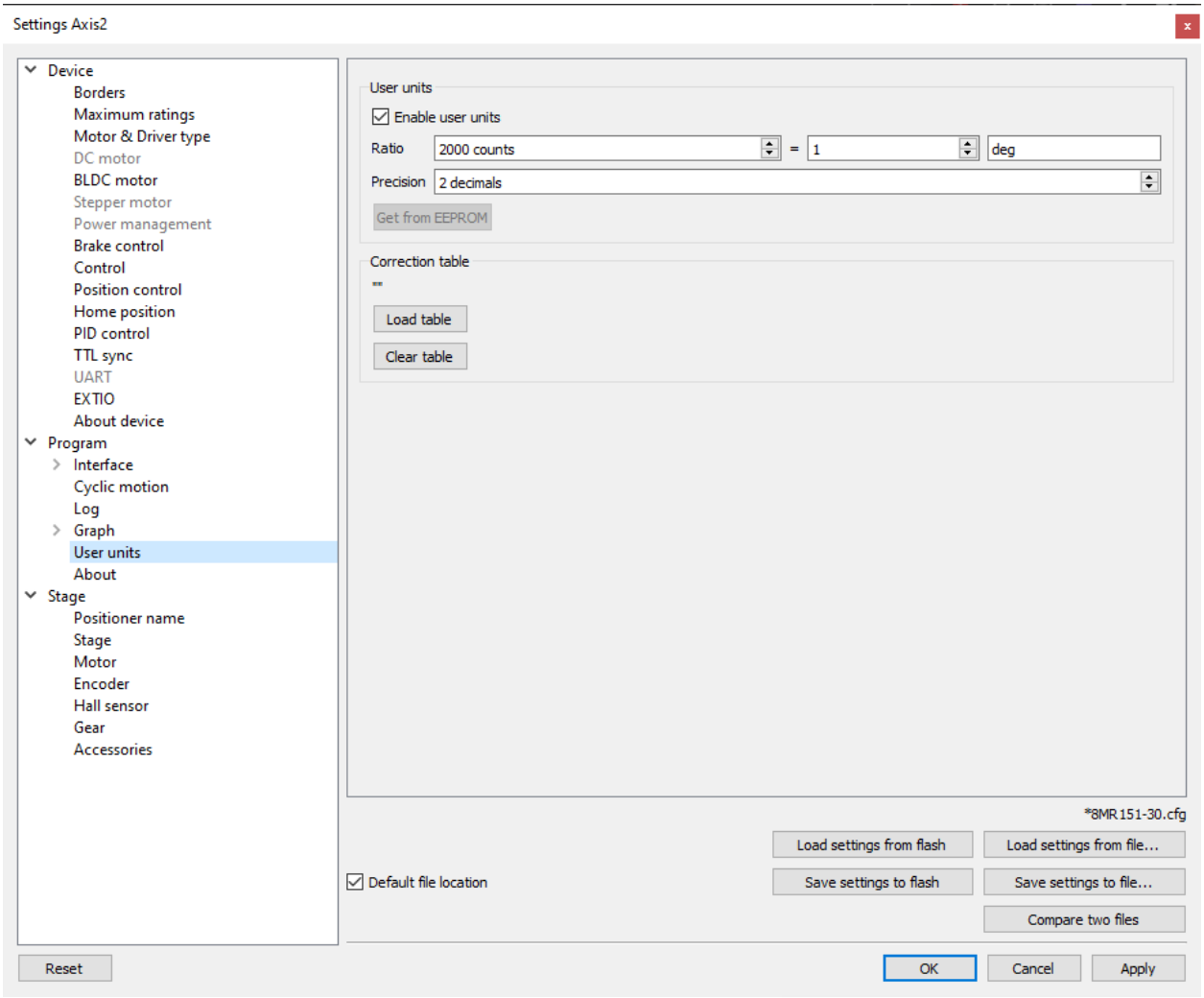


Fig. 5.45: User units tab

Use this tab to configure user units display. Used to replace internal controller coordinates with units familiar to the user. This tab also allows you to use the *coordinate correction table* for user units. The coordinate correction table allows you to significantly improve the positioning accuracy when using custom units.

5.4.8.1 User units

Enable user units - enables user unit display instead of steps (in case of stepper motor) or encoder counts (in case of DC motor). User units replace steps(counts) only in the main XILab window and do not affect any of the Settings pages.

Ratio - conversion of controller steps to position units, set as a ratio of two integer values “x steps = y user units”. Values “x”, “y” and unit name string are set by user.

Precision - displayed precision.

Get from EEPROM button reads user unit settings from connected EEPROM.

5.4.8.2 Coordinate correction table for more accurate positioning

Some functions for working with user units allow you to use the coordinate correction table for more precise positioning.

Load table - loads the *coordinate correction table*. If it is successfully loaded, the file name of the loaded table appears to the right of the button. From this moment certain _calb functions that perform the re-count procedure of coordinates using the correction table will proceed the recalculation, up to the clearance of the table with *Clear table*.

Clear table - clears the correction table. All _calb functions operate in a normal mode.

5.4.9 About the application

Program -> About in the *Application settings*

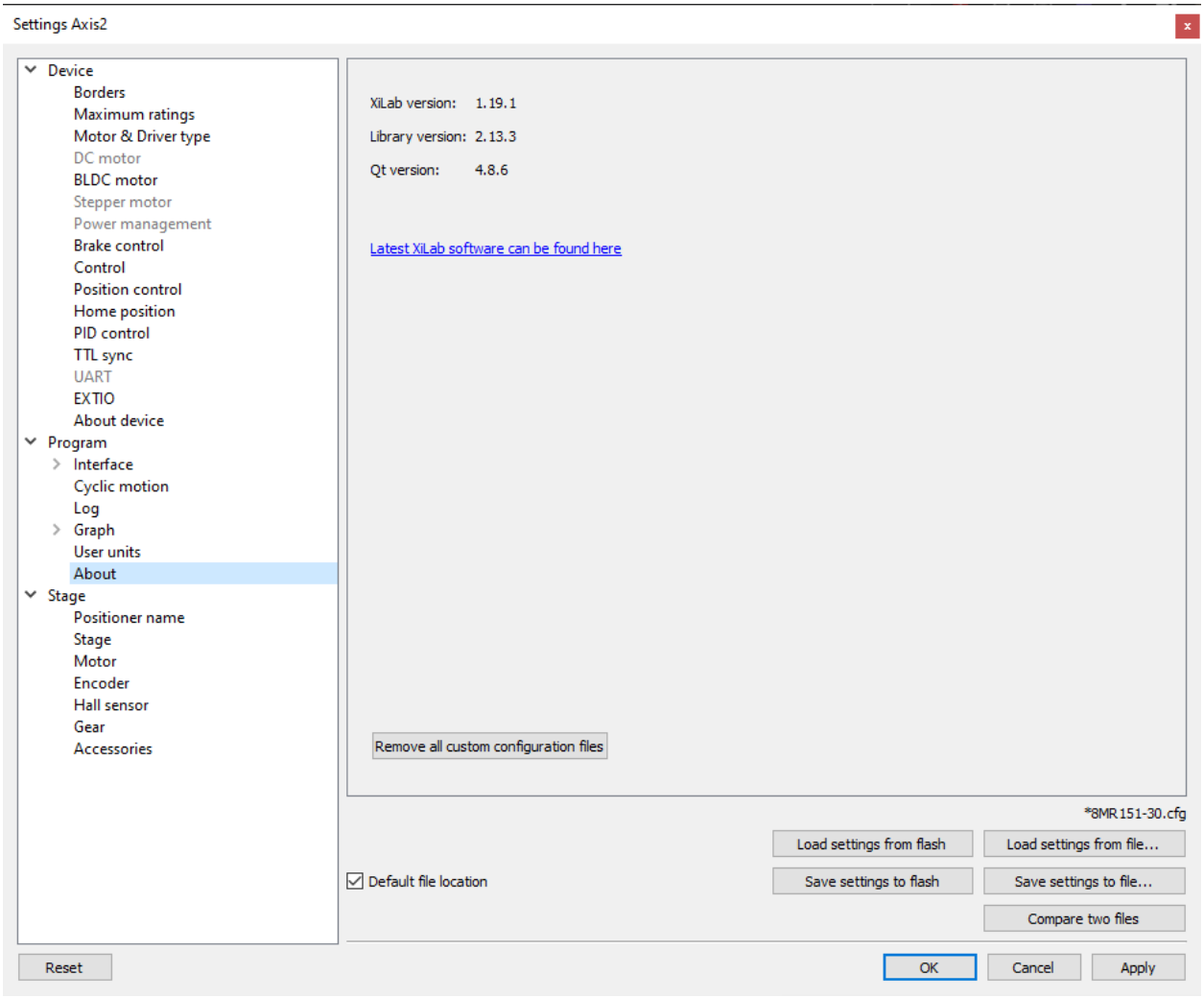


Fig. 5.46: About tab

This section displays the XILab application version. It also contains a link to the page with the latest Software version.

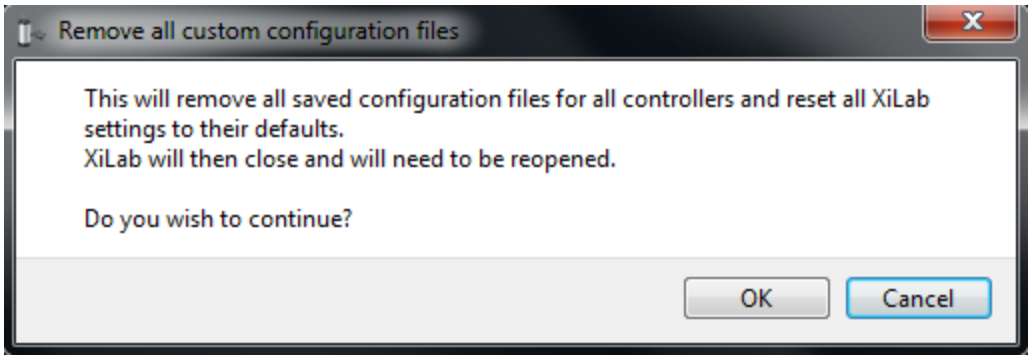


Fig. 5.47: User configuration file cleanup dialog

"Remove all custom configuration files" button displays a dialog prompt to delete all custom configuration files created by XiLab. Files to be deleted are located in XiLab configuration directory. These files are "settings.ini", which stores common program settings, "SNnnn.cfg", which store per-controller settings, "V_nnn", which store virtual controller internal states, "scratch.txt", which stores last run *script*. Here "nnn" means any number. Pressing OK in this dialog will delete all these files and close XiLab, pressing Cancel will abort deletion and close this dialog.

5.5 Stage specifications

5.5.1 Stage name

Stage -> stage name in the *Application settings*

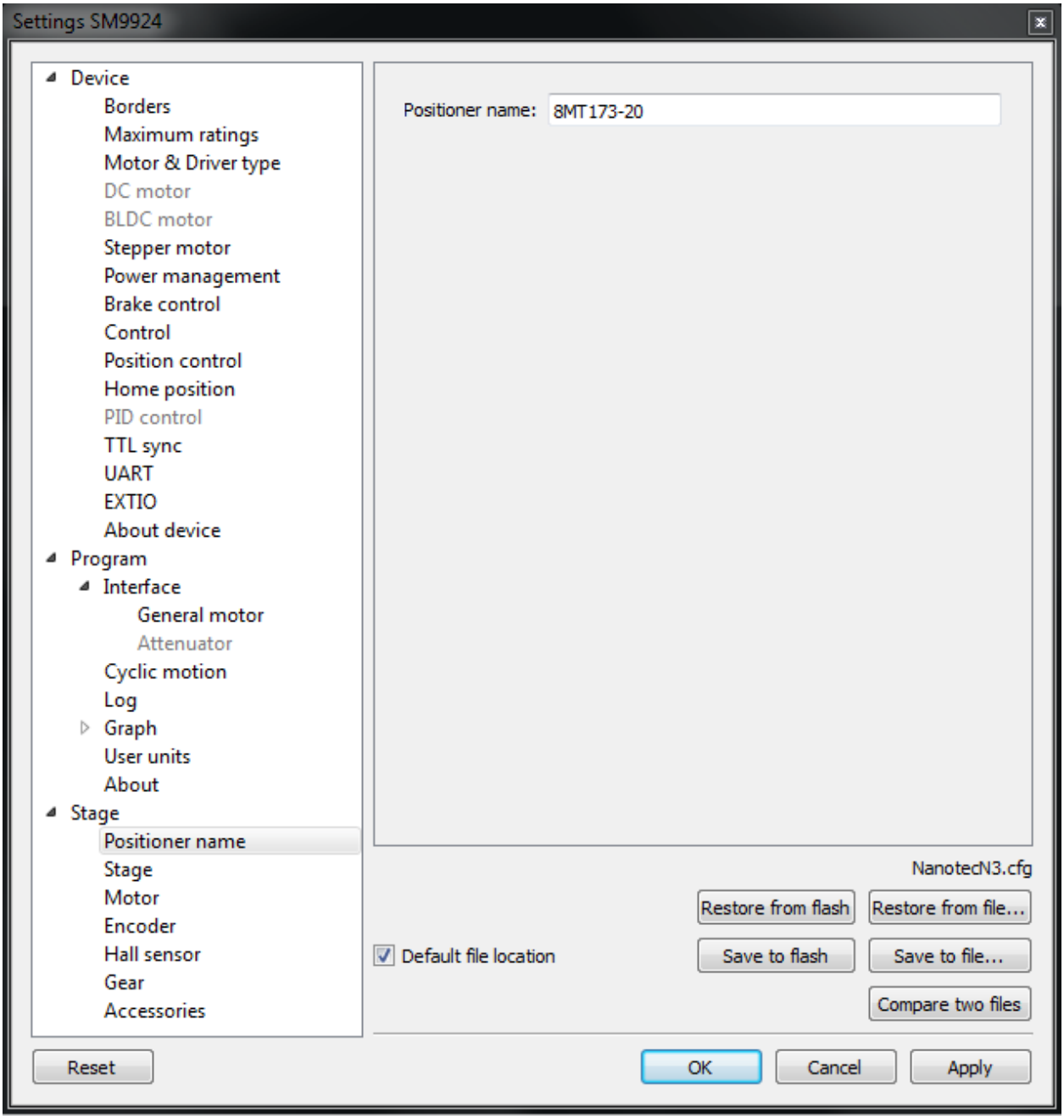


Fig. 5.48: Stage name window

This block contains the stage name (defined by user).

5.5.2 Stage general characteristics

Important: The information on the *"Stage -> Stage"* tab temporarily not used

Stage -> Stage in the *Application settings*

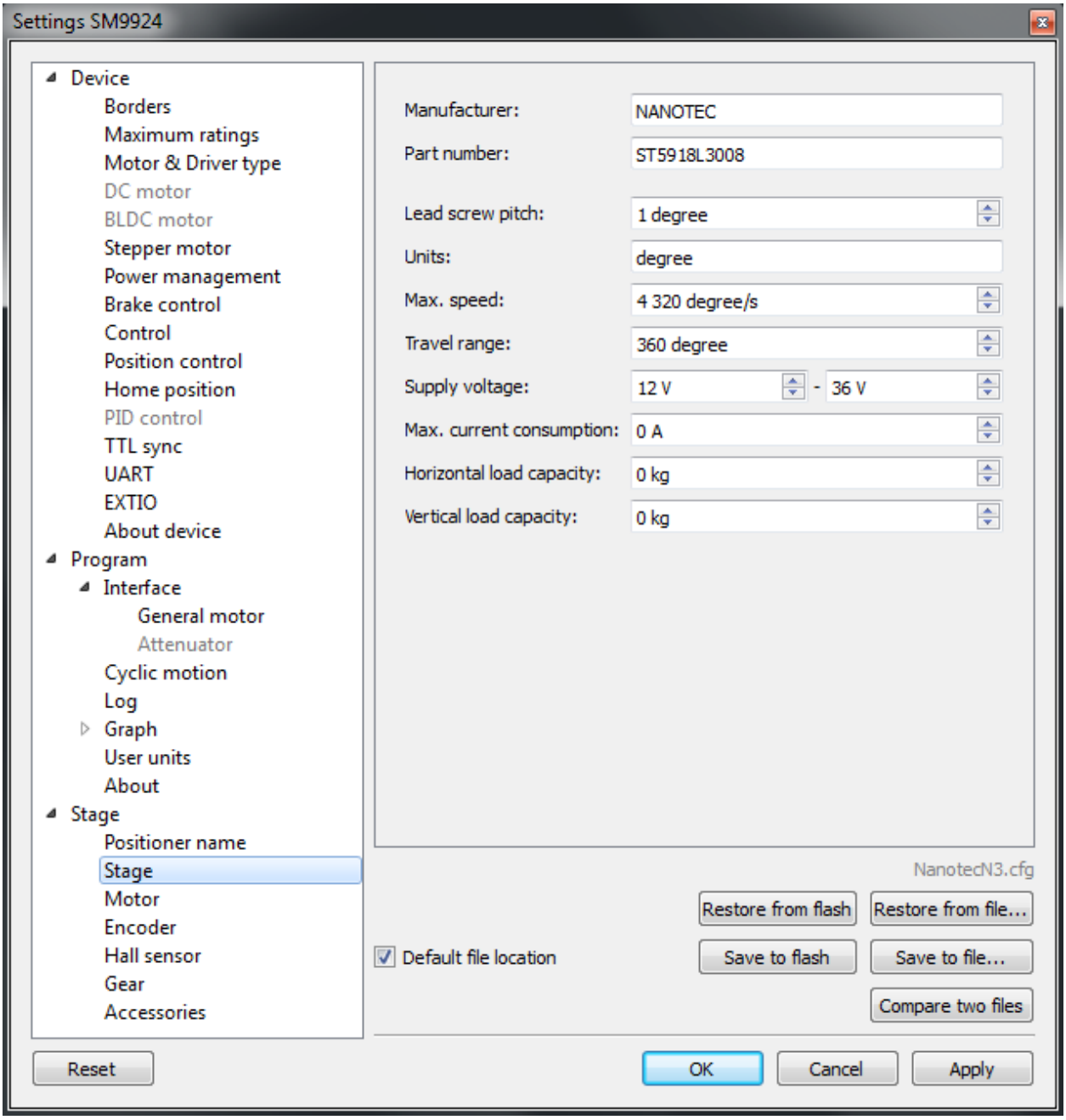


Fig. 5.49: Stage general characteristics window

Stage parameter prouop contains information about the stage:

- *Manufacturer* - the manufacturer name.
- *Part number* - the catalog number.
- *Lead screw pitch* - lead screw pitch.
- *Units* - stage movement measurement units (mm, degrees, steps).
- *Max. speed* - the maximum speed.

- *Travel range* - range of motion.
- *Supply voltage* - acceptable supply voltage range.
- *Max. current consumption* - the maximum current consumption.
- *Horizontal load capacity* - the maximum horizontal load on the stage.
- *Vertical load capacity* - the maximum vertical load on the stage.

5.5.3 Motor characteristics

Important: The information on the *"Stage -> Motor"* tab temporarily not used

Stage -> Motor in the *Application settings*

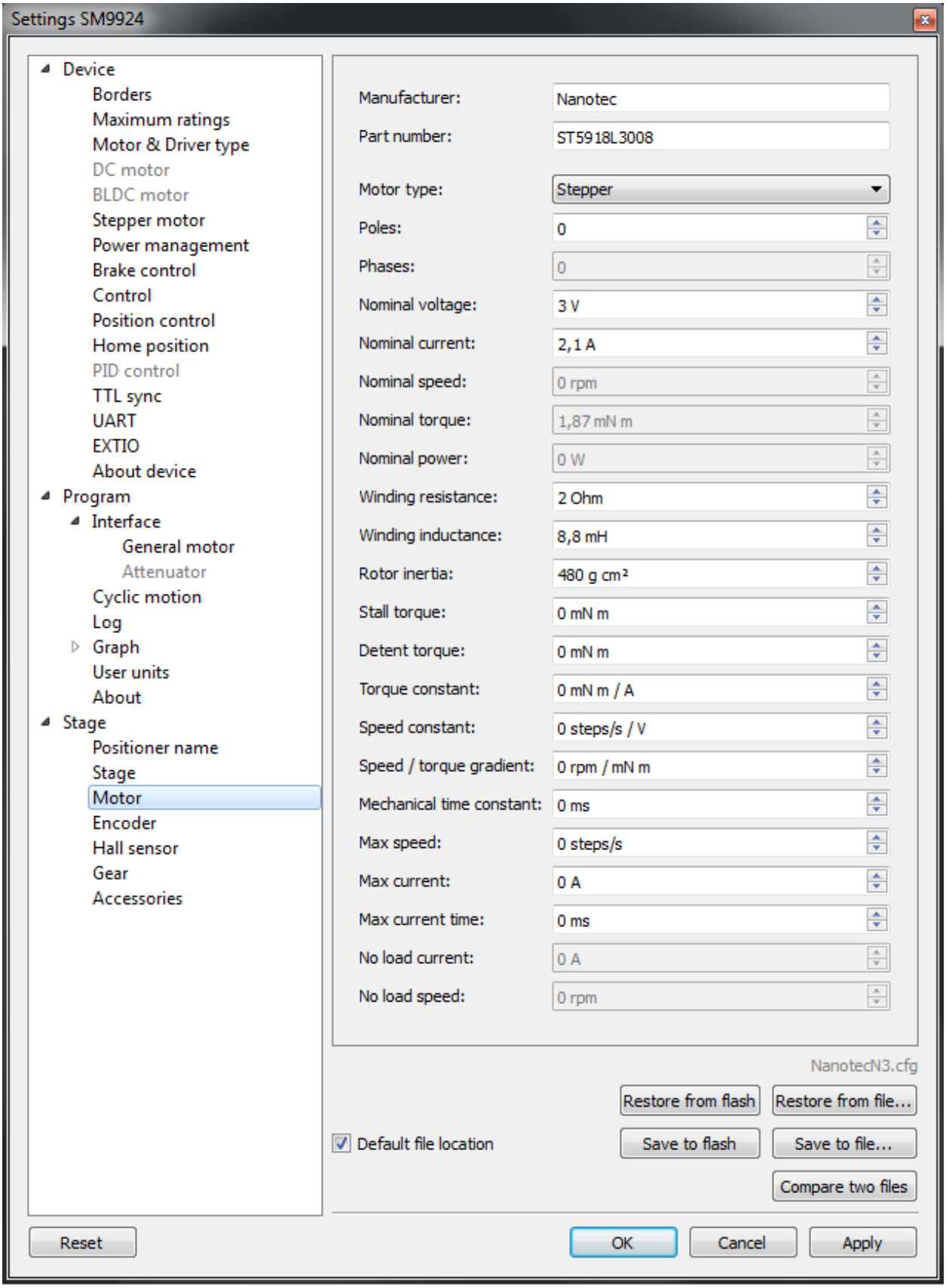


Fig. 5.50: Motor characteristics window

This section contains motor information:

- *Manufacturer* - motor manufacturer.
- *Part number* - catalog number.
- *Motor type* - motor type (stepper, DC or BLDC)
- *Poles* - number of pole pairs for DC or BLDC motors, steps per revolution for stepper motors.
- *Phases* - BLDC motor phases.
- *Nominal voltage* - nominal winding voltage.
- *Nominal current* - maximum continuous winding current for DC or BLDC motors, nominal current for stepper motors.
- *Nominal speed* - nominal speed.
- *Nominal torque* - nominal torque.
- *Nominal power* - nominal power consumption.
- *Winding resistance* - active resistance of the winding.
- *Winding inductance* - inductance of the winding.
- *Rotor inertia* - rotor inertia.
- *Stall torque* - zero speed torque.
- *Detent torque* - hold torque with unpowered windings.
- *Torque constant* - torque constant.
- *Speed constant* - speed constant.
- *Speed/torque gradient* - speed/torque constant.
- *Mechanical time constant* - motor time constant.
- *Max speed* - maximum allowed speed.
- *Max current* - maximum allowed winding current.
- *Max current time* - maximum safe time of max current in the winding.
- *No load current* - no-load current consumption.
- *No load speed* - no-load motor speed.

5.5.4 Encoder specifications

Important: The information on the *"Stage -> Encoder"* tab temporarily not used

Stage -> Encoder in the *Application settings*

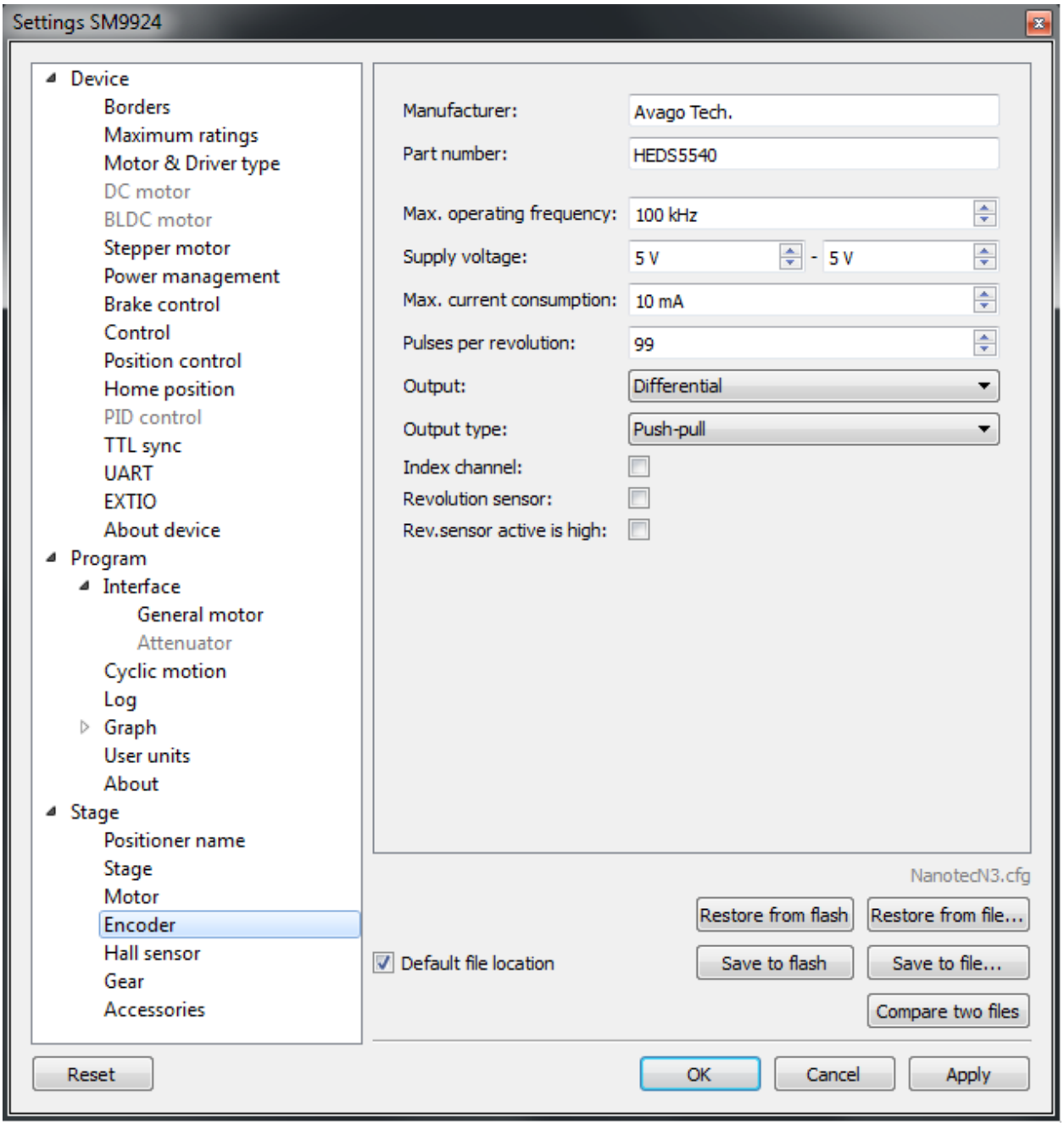


Fig. 5.51: Window Encoder specifications

This section contains information about *encoder*:

- *Manufacturer* - encoder manufacturer name.
- *Part number* - the catalog number.
- *Max. operating frequency* - the maximum operating frequency.
- *Supply voltage* - acceptable supply voltage range.
- *Max. current consumption* - the maximum current consumption.

- *Pulses per revolution* - pulses per single motor shaft revolution.
- *Output* - differential or single-ended output.
- *Output type* - electric output type (push-pull or open drain).
- *Index channel* - indicates if additional index channel is present.
- *Revolution sensor* - indicates if revolution sensor is present.
- *Rev.sensor active is high* - if enabled then active revolution sensor state is logical 1, else it is logical 0.

5.5.5 Hall sensor characteristics

Important: The information on the *"Stage -> Hall sensor"* tab temporarily not used

Stage -> Hall sensor in the *Application settings*

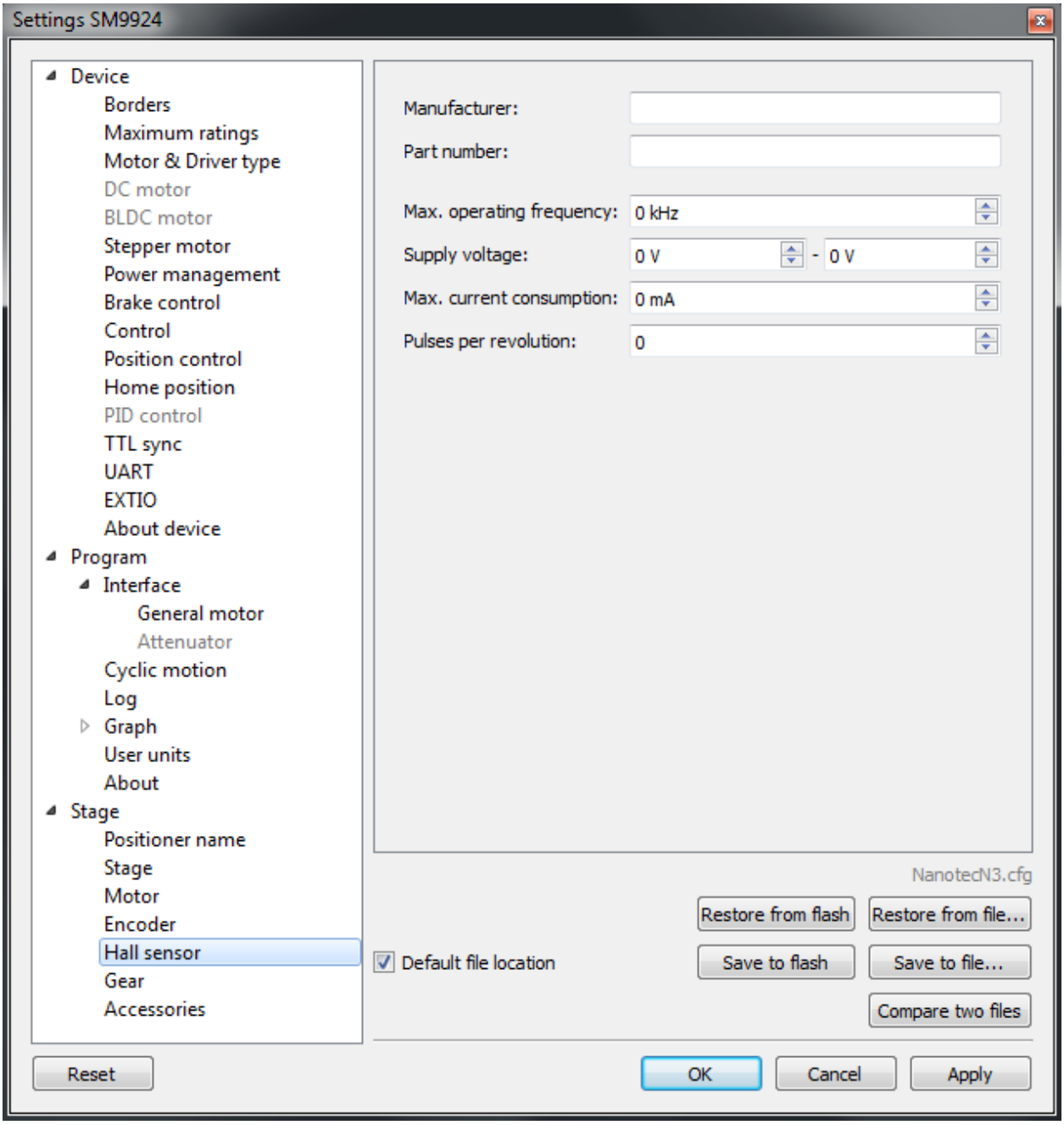


Fig. 5.52: Hall sensor properties window

This section contains information about hall sensor:

- *Manufacturer* - sensor manufacturer name.
- *Part number* - the catalog number.
- *Max. operating frequency* - the maximum operating frequency.
- *Supply voltage* - acceptable supply voltage range.
- *Max. current consumption* - the maximum current consumption.

- *Pulses per revolution* - pulses per single motor shaft revolution.

5.5.6 Reducing gear specifications

Important: The information on the *"Stage -> Gear"* tab temporarily not used

Stage -> Gear in the *Application settings*

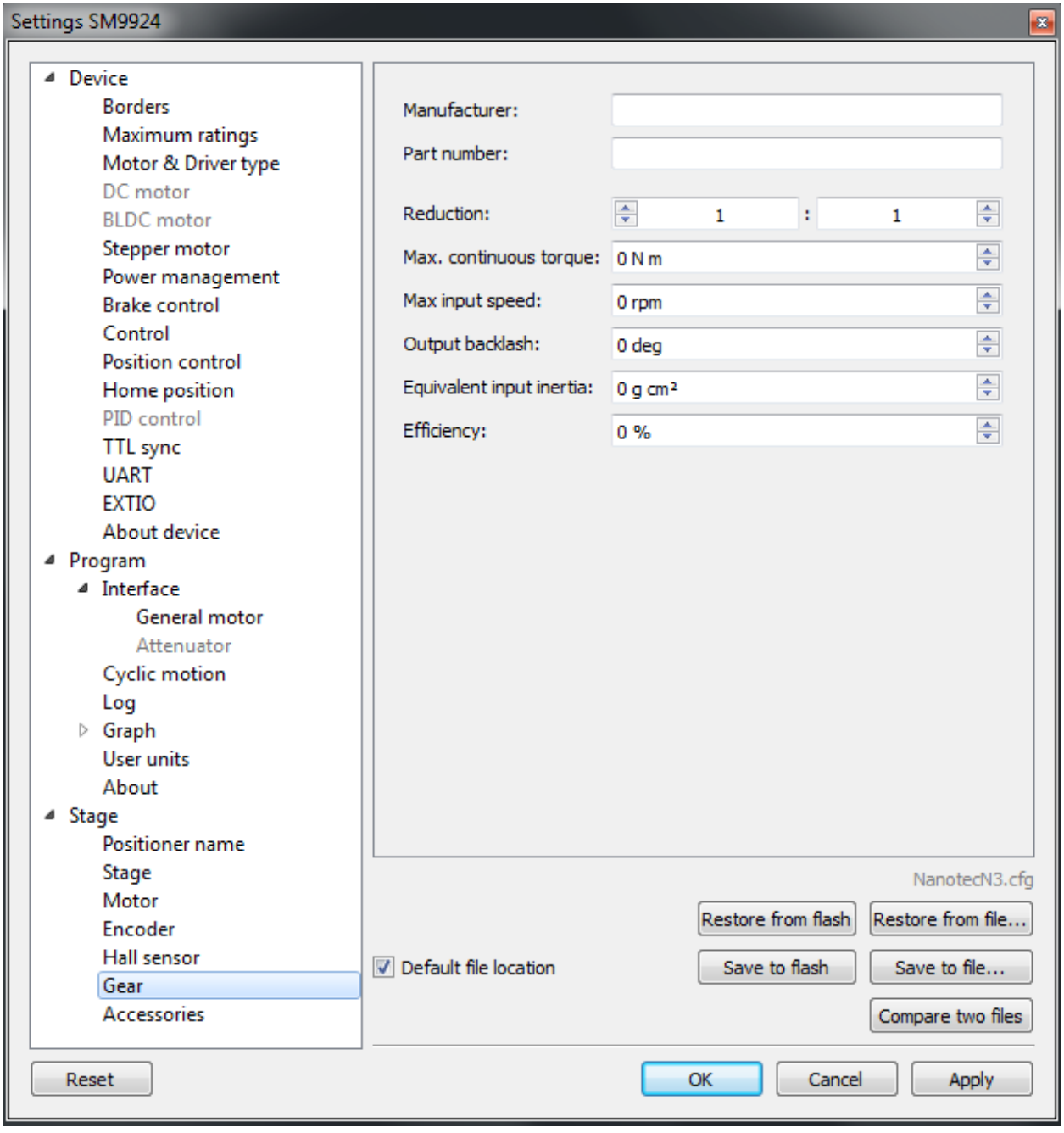


Fig. 5.53: Reducing gear specifications window

This section contains information about the reducing gear:

- *Manufacturer* - the manufacturer name.
- *Part number* - the catalog number.
- *Reduction* - gear transmission ratio.
- *Max. continious torgue* - maximum torque at the gear input.
- *Max. input speed* - the maximum speed at the gear input.
- *Output backlash* - gear output backlash.
- *Equivalent input inertia* - equivalent input inertia of the gear.
- *Efficiency* - Gear efficiency.

5.5.7 Accessories specifications

Important: The information on the *"Stage -> Accessories"* tab temporarily not used

Stage -> Accessories in the *Application settings*

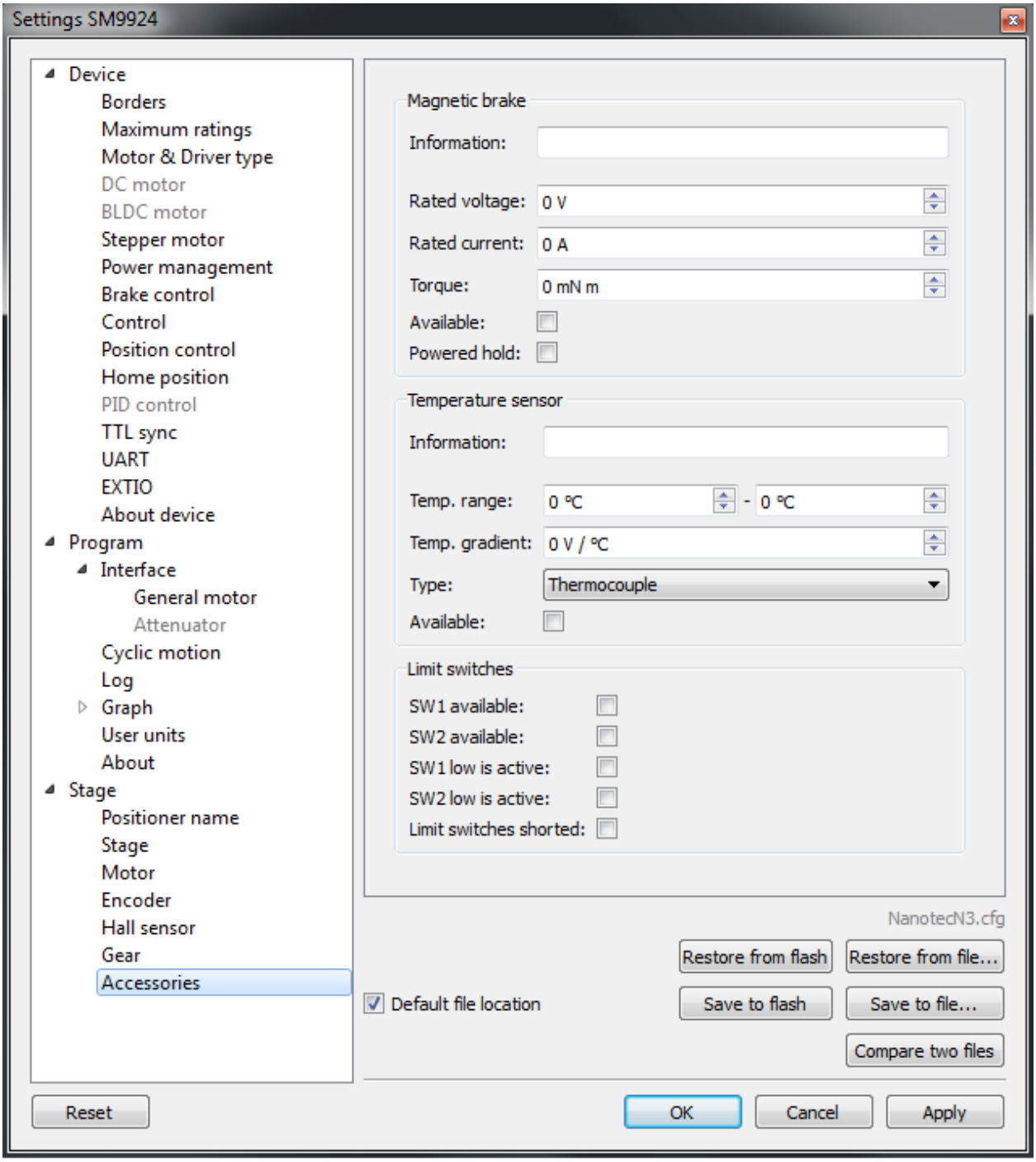


Fig. 5.54: Accessories specifications window

This panel contains information about various accessories.

Magnetic brake - magnetic brake section:

- *Information* - magnetic brake manufacturer and part number.
- *Rated voltage* - nominal magnetic brake voltage.

- *Rated current* - nominal magnetic brake current.
- *Torque* - hold torque.
- *Available* - indicates if magnetic brake is available.
- *Powered hold* - magnetic brake is in hold mode on power on if this option is enabled.

Temperature sensor - temperature sensor section:

- *Information* - temperature sensor manufacturer and part number.
- *Temp. range* - measuered temperature range.
- *Temp. gradient* - temperature gradient.
- *Type* - sensor type (thermocouple or semiconductor).
- *Available* - indicates if temperature sensor is available.

Limit switches - limit switches section:

- *SW1 available* - indicates if SW1 limit switch is available.
- *SW2 available* - indicates if SW2 limit switch is available.
- *SW1 low is active* - indicates if SW1 limit switch active state is low.
- *SW2 low is active* - indicates if SW2 limit switch active state is low.
- *Limit switches shorted* - if enabled, then limit switches are shorted.

5.6 Correct shutdown

Correct shutdown assumes shutdown of the motor and saving the current position by the controller. The current position is automatically saved, see *Saving the position in FRAM memory*.

Exit button performs correct shutdown and exit. When you click it the application sends a soft stop command to the controller, and after the stop is complete, the application sends command of power-off. If execution of the soft stop command was interrupted by an event like a motion command from the *joystick* or signal of *TTL synchronization*, or if while sending of soft stop command or command of power off to the controller, the library returned an error, the exit will be canceled. In this case you need to check *joystick settings and settings of "right" and "left" buttons* and *Synchronization settings*.

5.7 Working over network

Note: The **8SMC4-USB-Eth** adapter has been renamed **8Eth1**[1]. All available instructions in our manual are relevant and applicable to both devices.

XiLab is capable of interacting with remote controllers via Ethernet. However you need a special XIMC server, which can be either obtained as a standalone program and installed on any appropriate device, or as one of the services of *8Eth1* adapter.

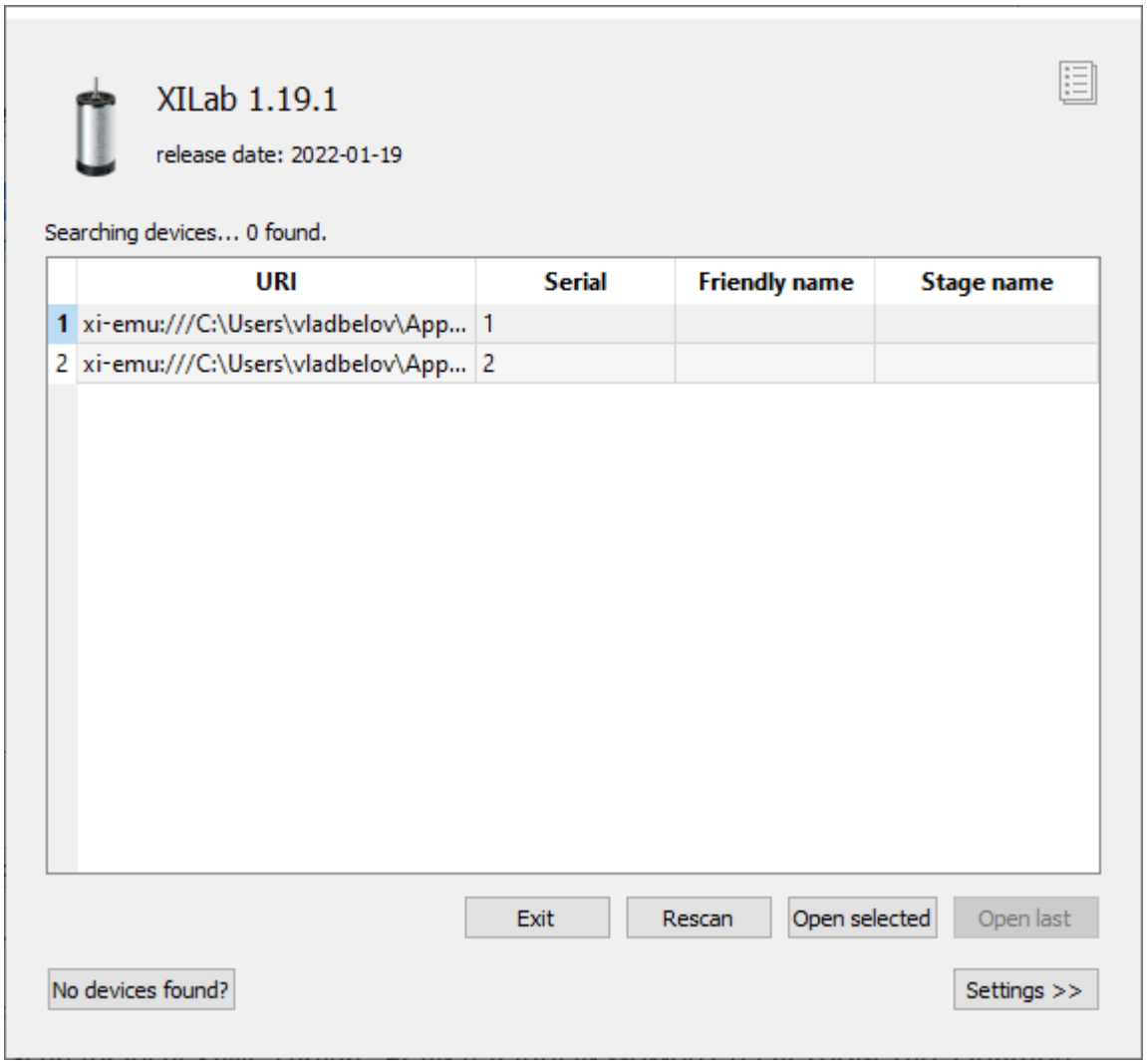
5.7.1 Getting started

- Before launching, connect controllers to the device with XIMC server installed using USB cable. At the same time, it is assumed that controllers can be connected to the motor and energized from power supply (see *Overview and getting started* for more details).

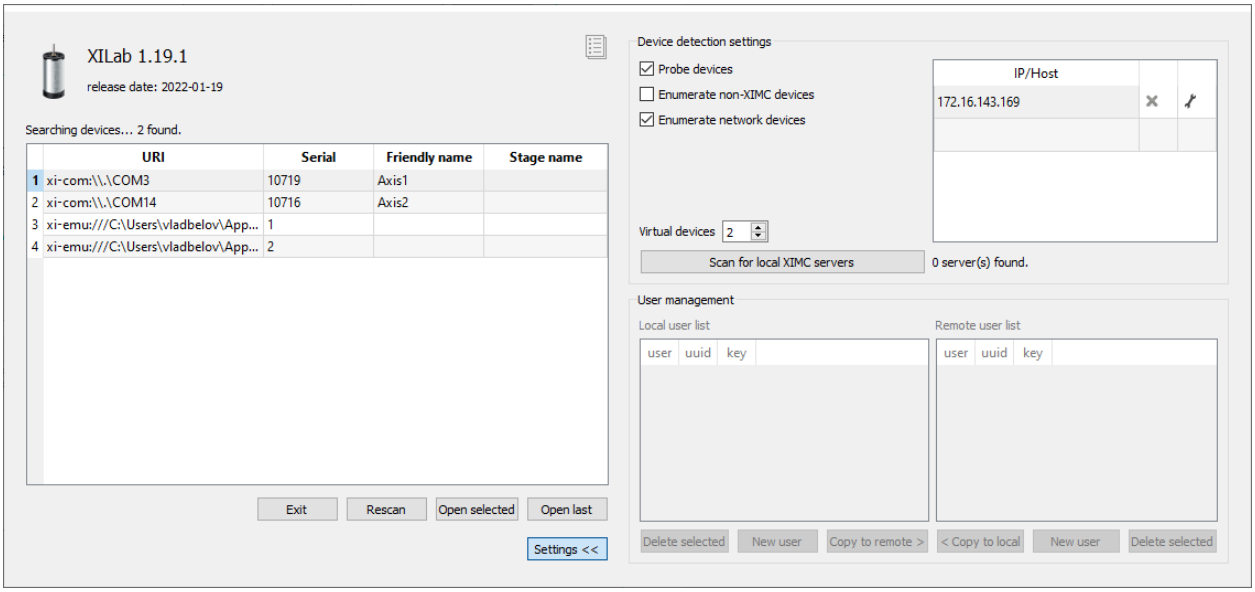
[1] The vendor code was changed in 2020, earlier this product had the vendor code 8SMC4-USB-Eth.

- Connect the device with XIMC server to the same subnet that contains the control computer, DHCP server. In case of *8Eth1* adapter supply it with power adapter (via 5V - 2A connector) and wait for a minute to finish Linux boot on the onboard computer.
- Launch XiLab and make the following.

At first start, XiLab opens controller detection window with two virtual devices.



Click *Settings*, check *Enumerate network devices* in the right tab and press *Scan for local XIMC servers* button. XiLab will use broadcast request to find available XIMC servers in your LAN and for each detected instance an entry in the "IP/hosts" list will be created. You can also edit address list manually if desired.



Click *Rescan* button in the left tap, XiLab will find all axis connected to the system. In controller detection window choose an axis you need. You can control it in *single-axis mode* or in *multi-axis mode* if more than one axis was chosen. For additional information please refer to *Getting started with XiLab software* and *XILab application User's guide*.

Note: Once the device IP address has been found, it should be understood that moving the device to another location may lead to a change in its IP.

Note: Working with multiple adapters may cause a problem when the same board responds in a broadcast requests. You can find a new device by two different ways :

- Disconnect other 8Eth1 adapters, find the device on the network, connect all again.
- Press *Scan for local XIMC servers* button until you find sought-for device.

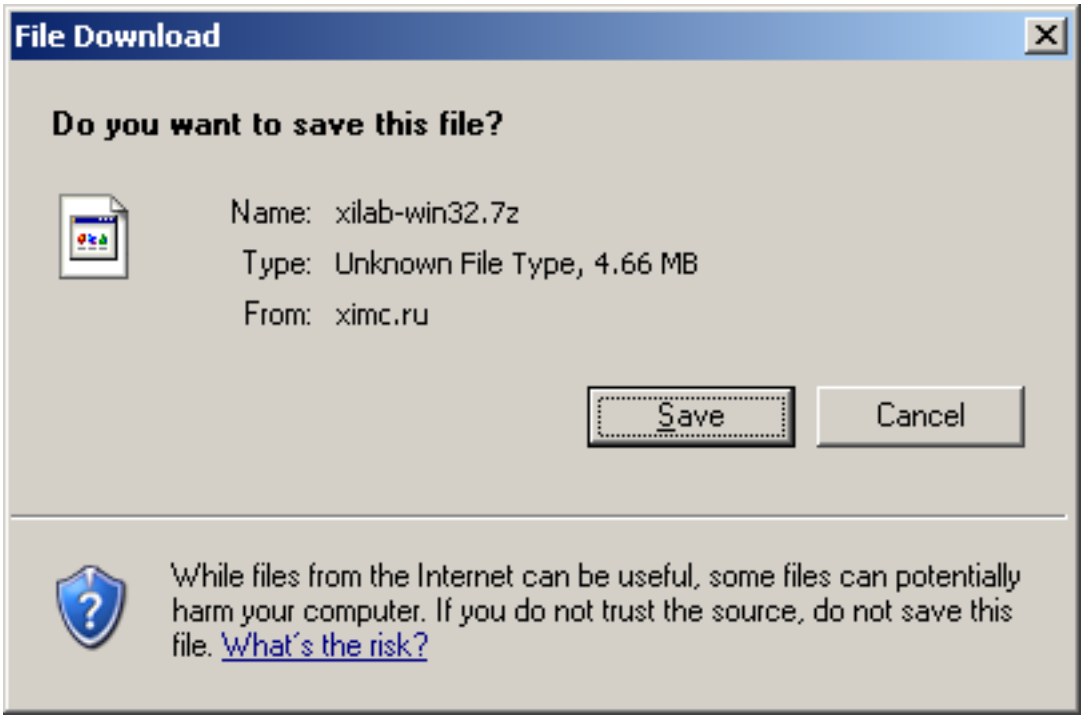
5.8 XILab installation

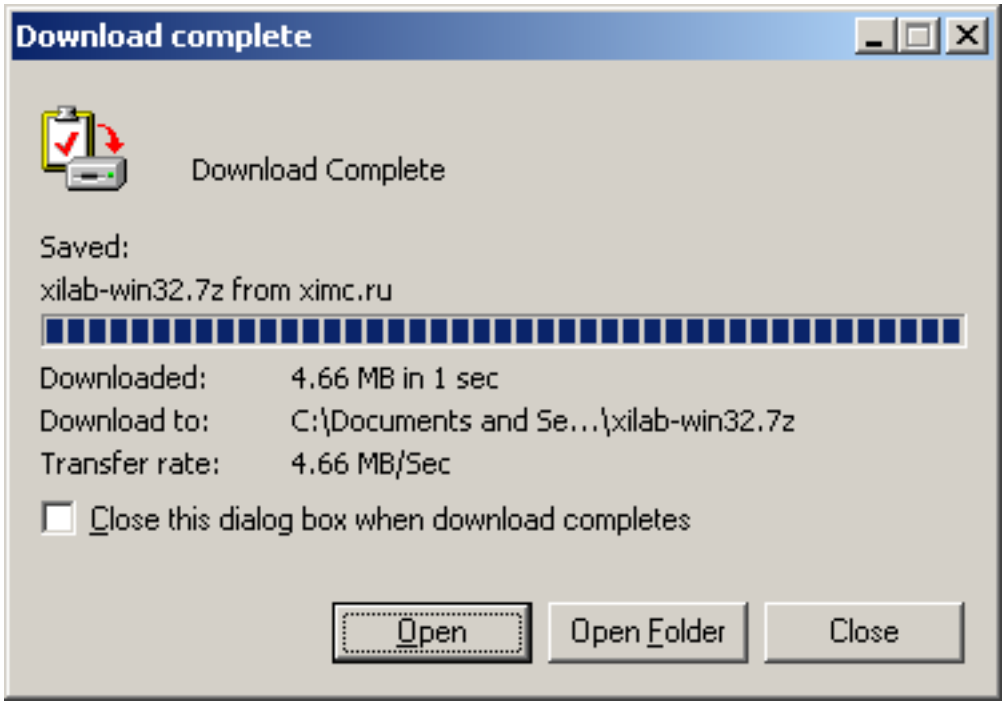
5.8.1 Installation on Windows

5.8.1.1 Installation on Windows XP

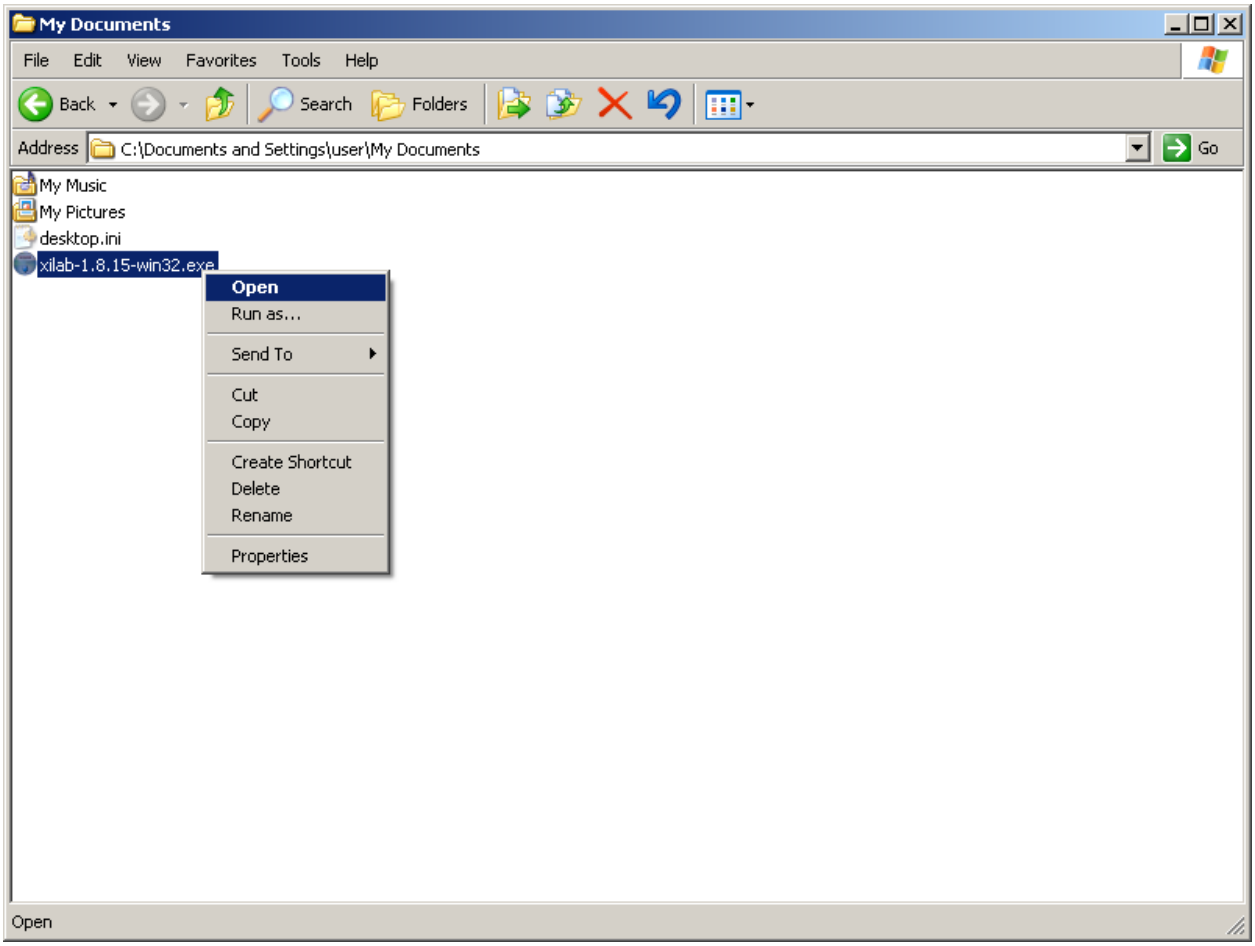
Copy the installer program file to your computer. The installer file name is "xilab-<version_name>.exe". It automatically detects whether it is running on 32-bit or 64-bit version of Windows and installs the appropriate version of XiLab.

Note: Only Windows XP SP3 is supported. Please update your Windows XP to the latest service pack.





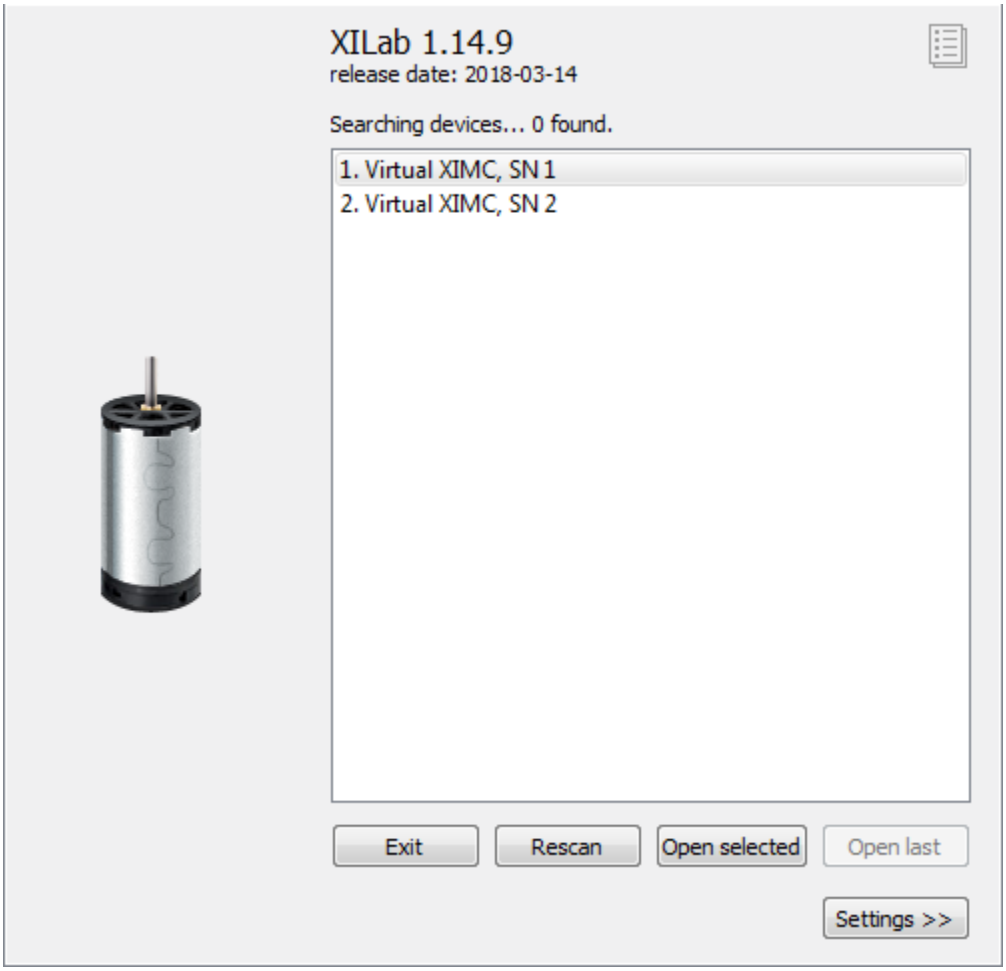
Run the installer and follow the on-screen instructions.



All the necessary software including drivers, packages and programs will be installed automatically.

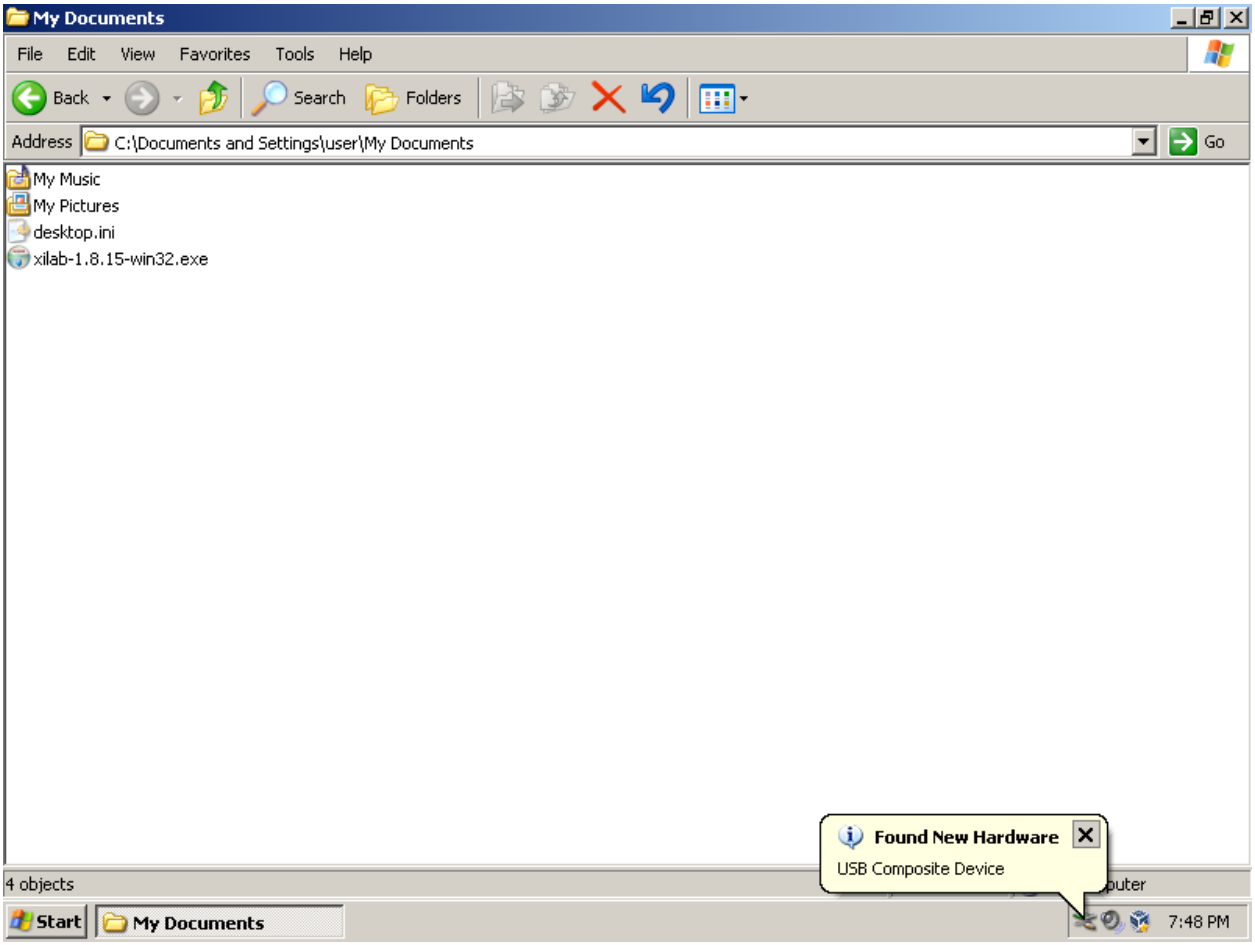


After the installation is complete the XiLab application will be started by default.



Connect the stage to the controller. Connect regulated power supply to the controller. Ground the controller or the power supply unit. Connect the controller to the computer using a USB-A - mini-USB-B cable.

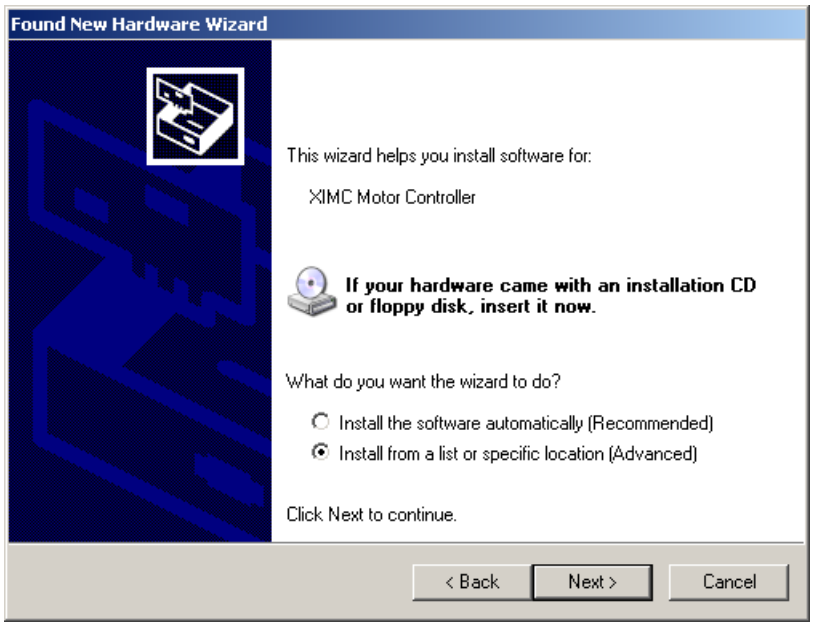
LED indicator on the controller board will start to flash. New Hardware Wizard will start after the first controller is connected to the computer. Wait until Windows detects the new device and installs the drivers for it.



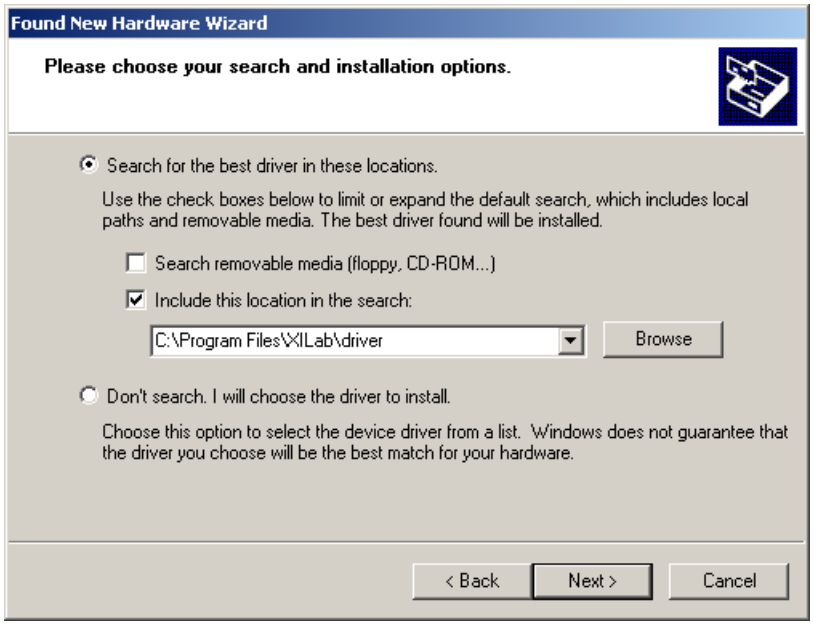
If the driver was not automatically installed select “No, not this time” and click “Next>” in the pop-up window.



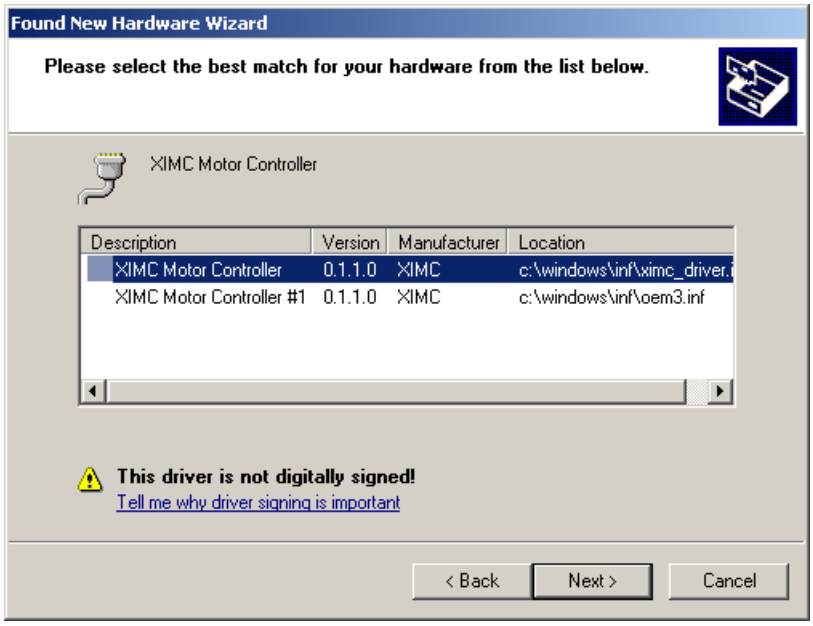
In the next window select “Install from a list or specific location (Advanced)” and click “Next>”.



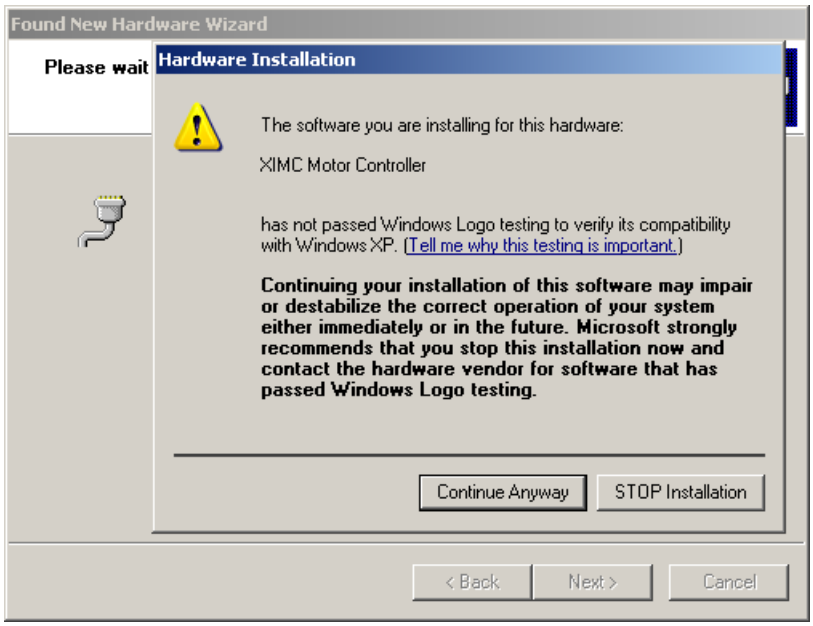
Select the *.inf file on the disk with the software supplied with the controller or in the program directory (by default the path is `C:\Program Files\XiLab\driver\`) and click “Next”.



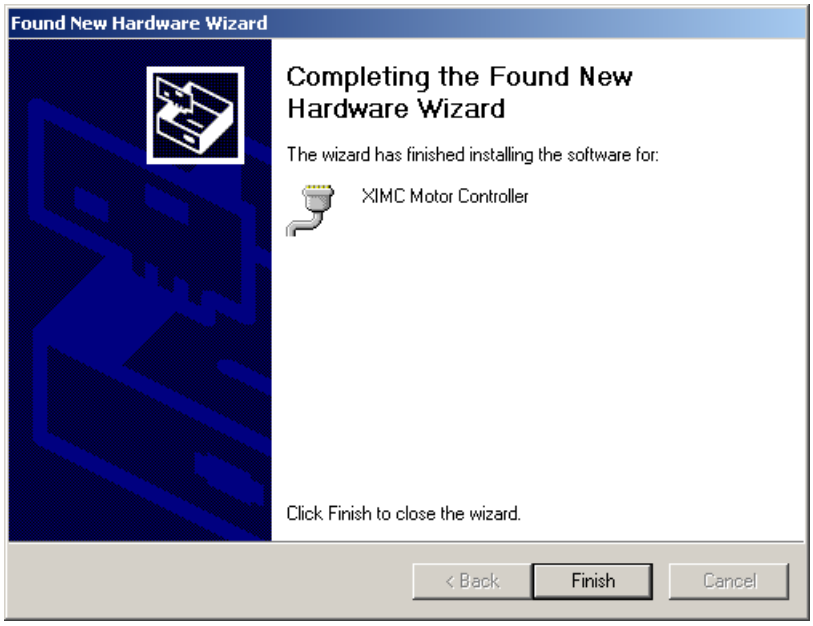
Click “Next”.



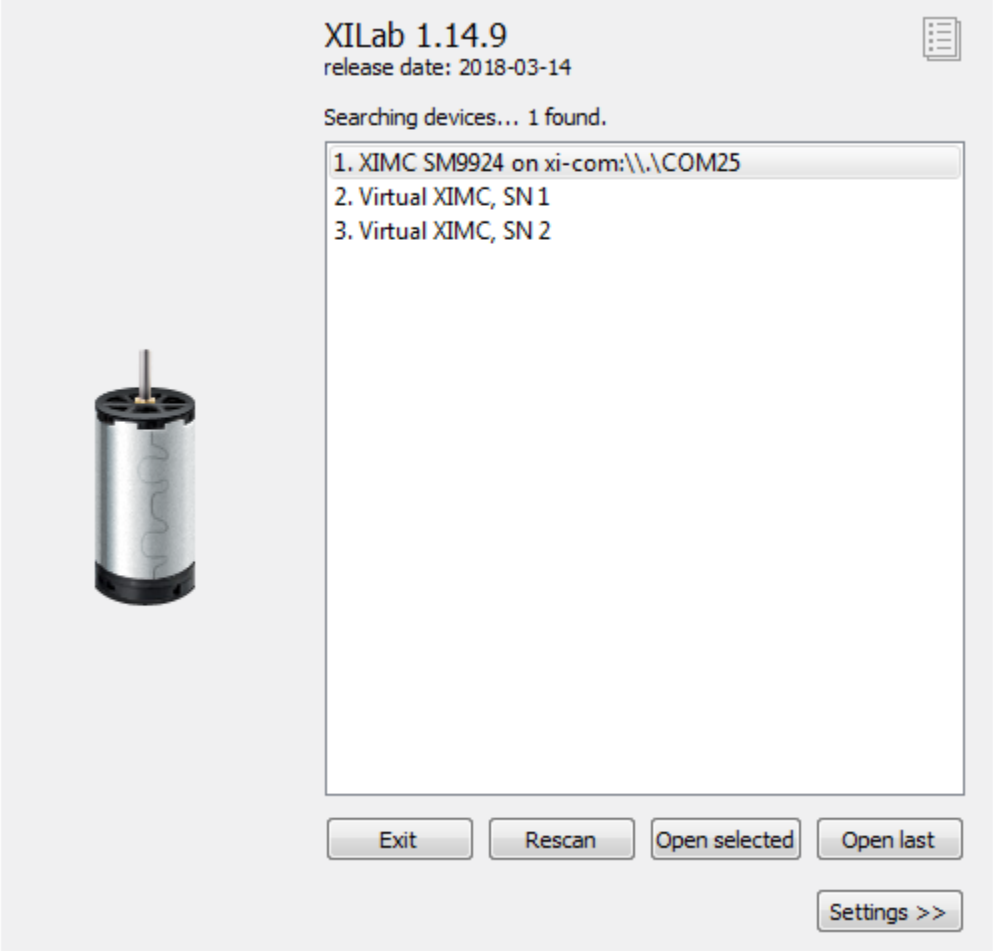
Click “Continue anyway”.

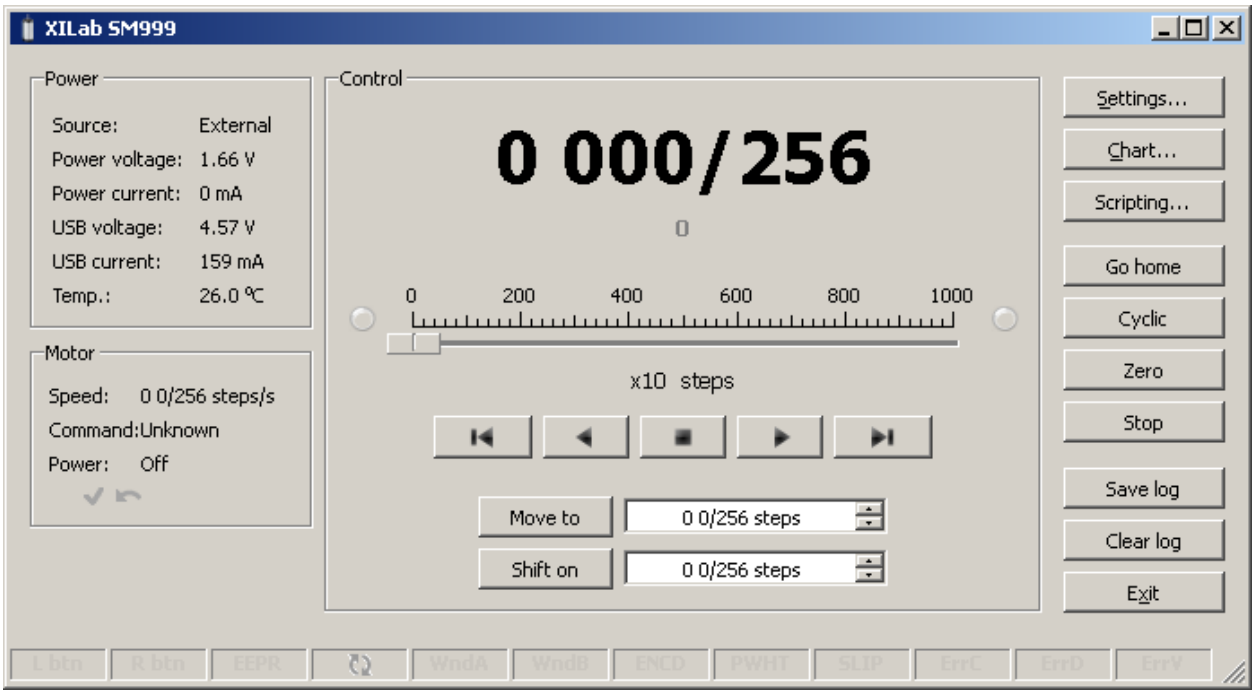


Click “Finish”. Driver installation is complete.



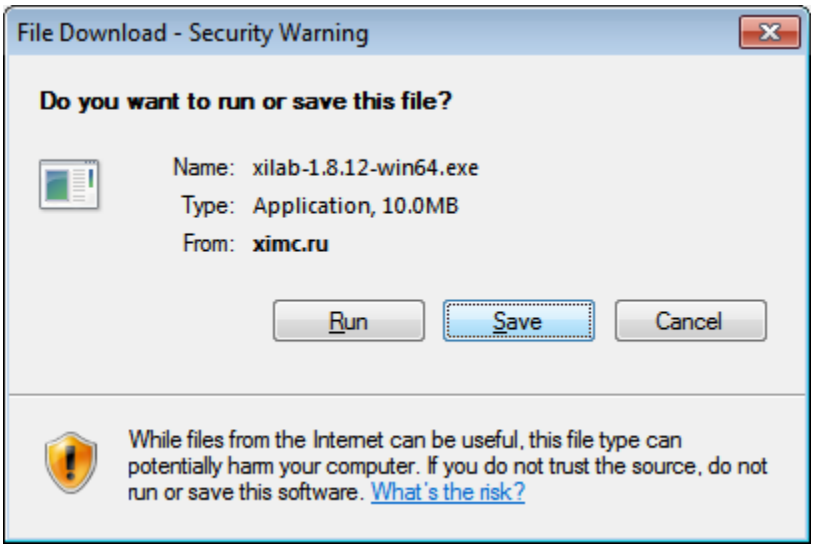
Click Retry or run the Xilab application again if it was closed. The system will detect the connected controller and open the main Xilab window.

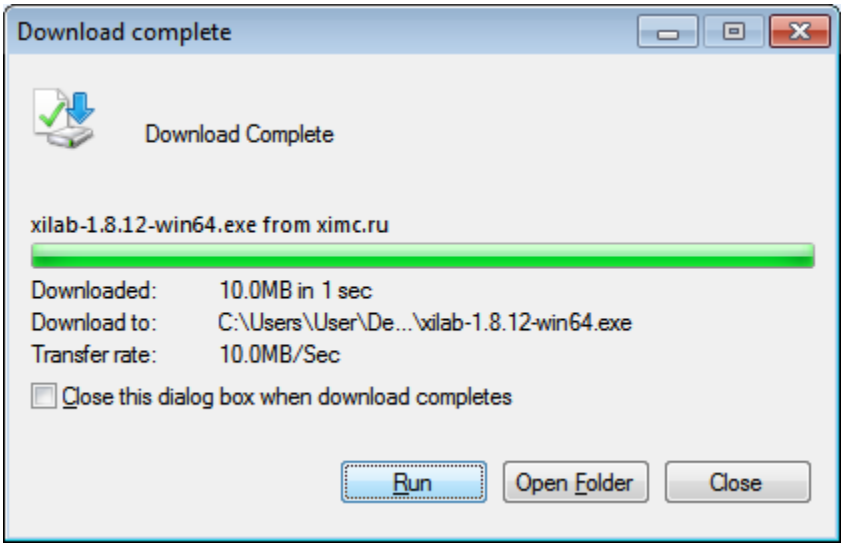




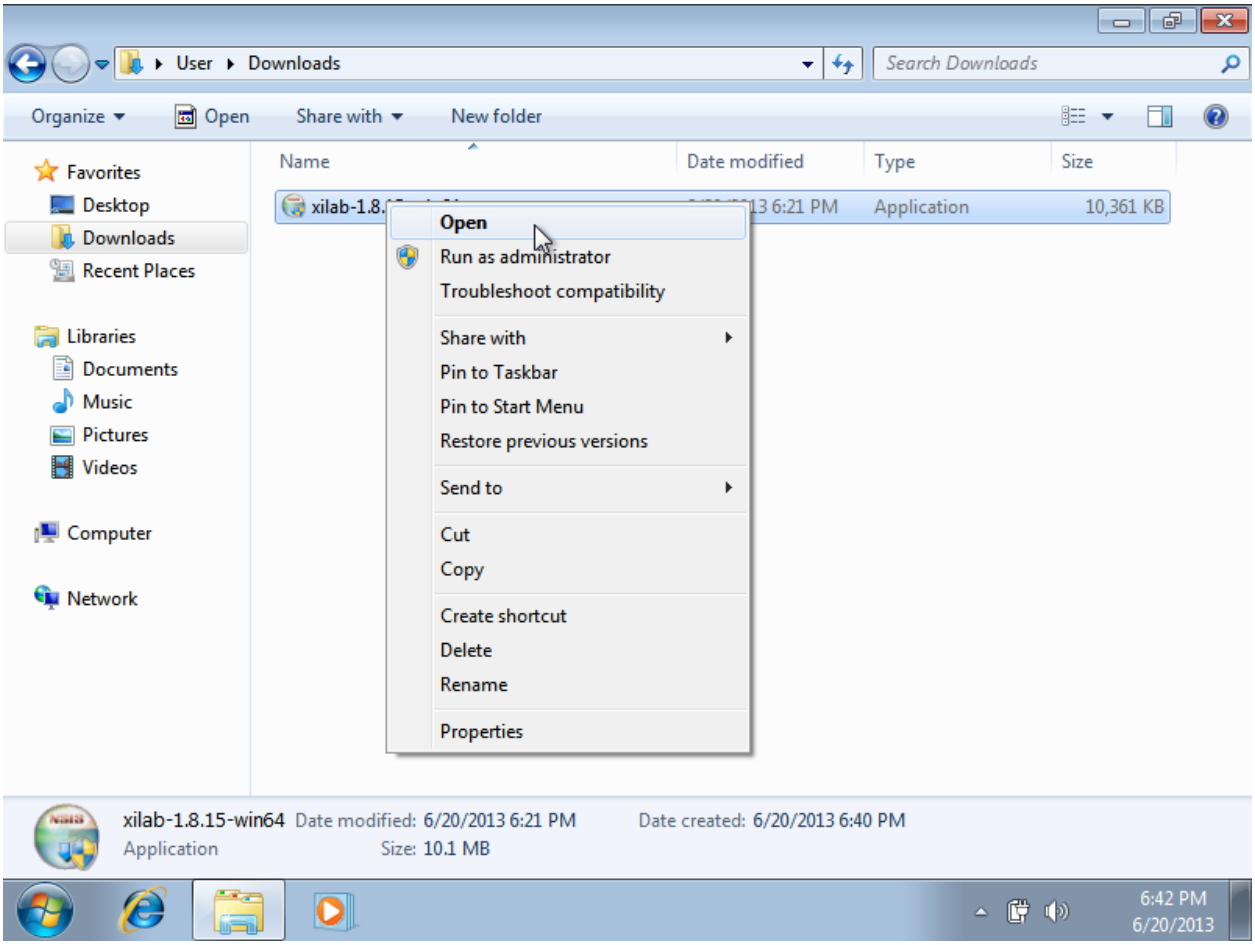
5.8.1.2 Installation on Windows 7

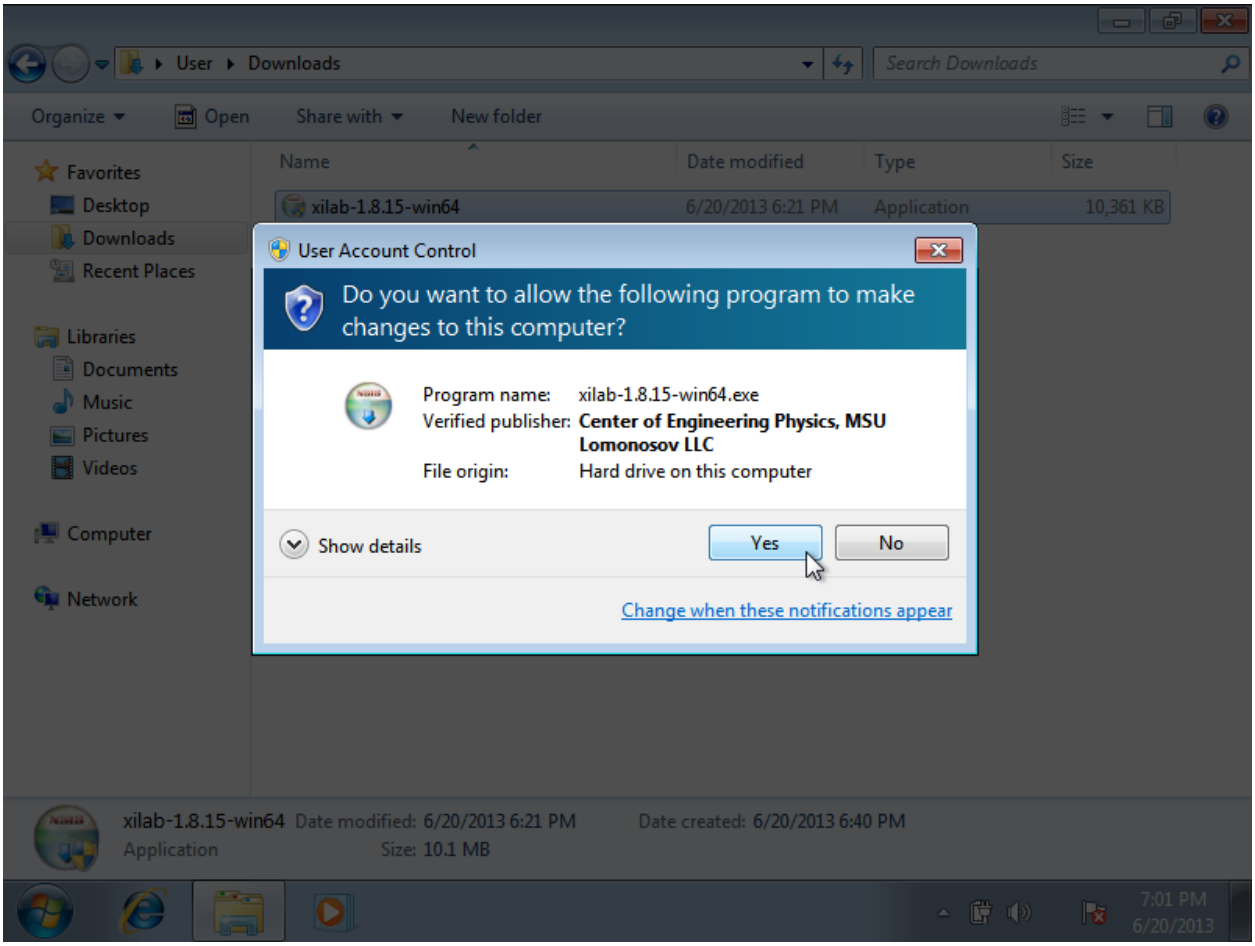
Copy the installer program file to your computer. The installer file name is "xilab-<version_name>.exe". It automatically detects whether it is running on 32-bit or 64-bit version of Windows and installs the appropriate version of XiLab.





Run the installer and follow the on-screen instructions.

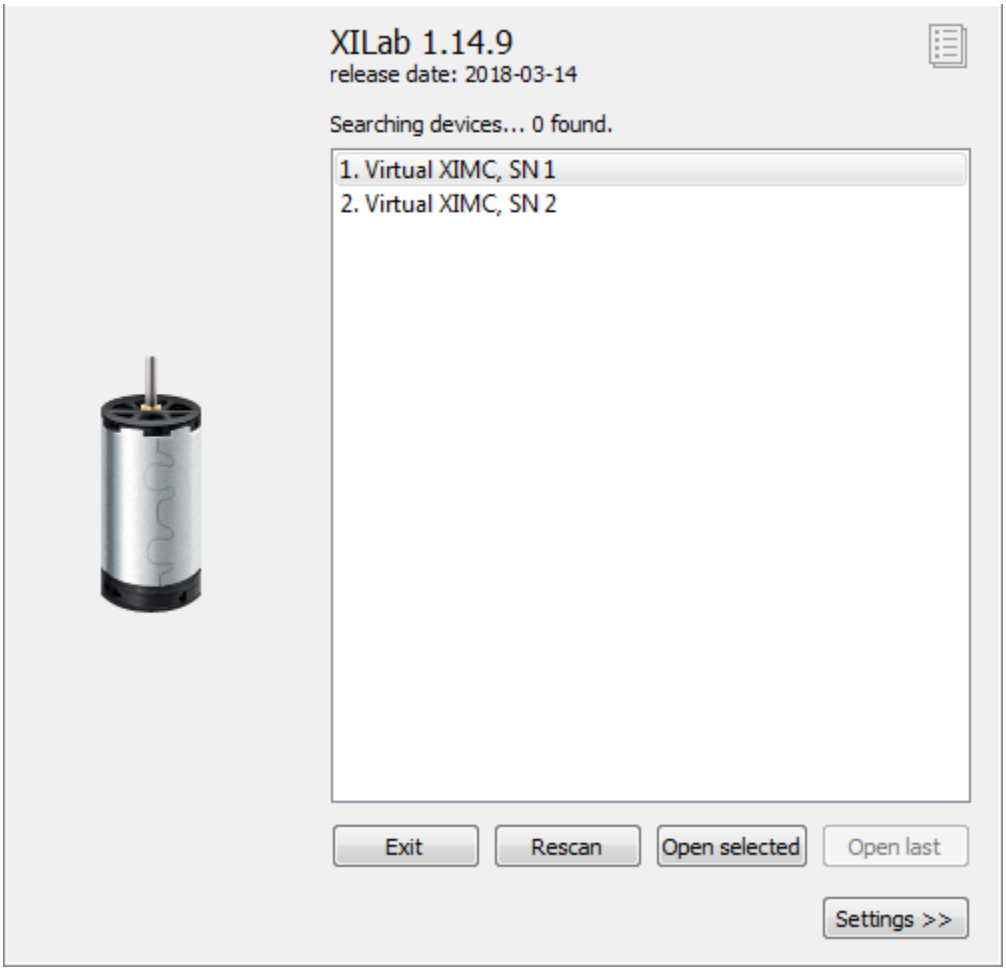




All the necessary software including drivers, packages and programs will be installed automatically.

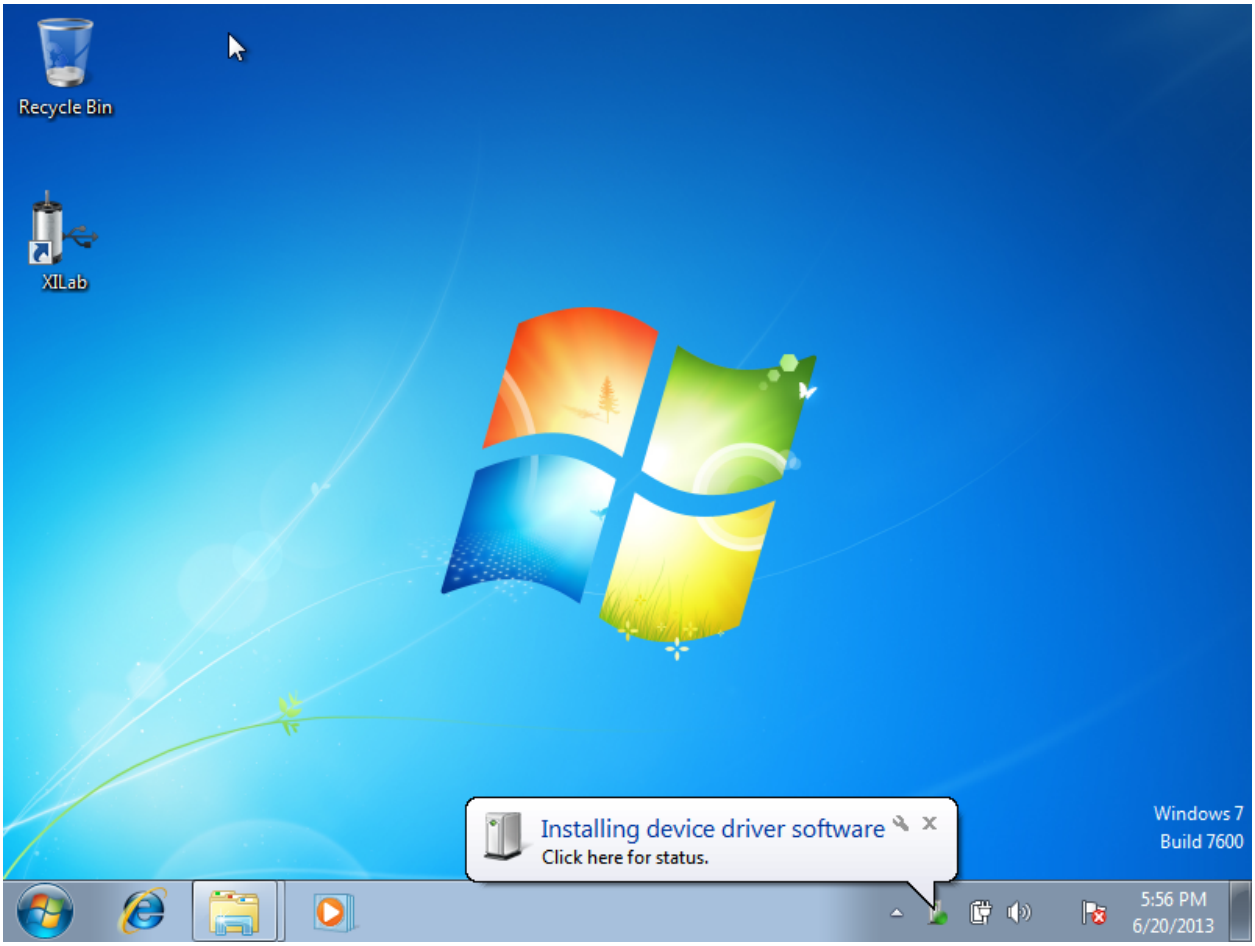


After the installation is complete the XiLab application will be started by default.

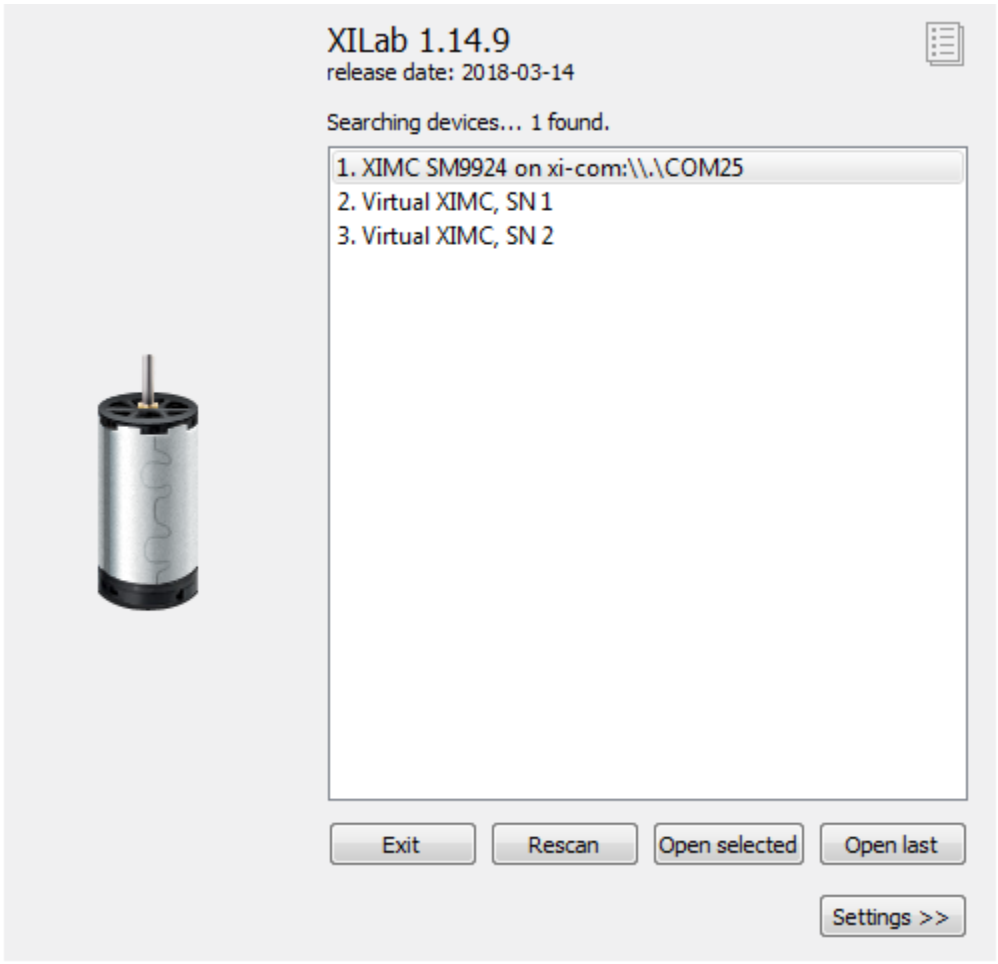


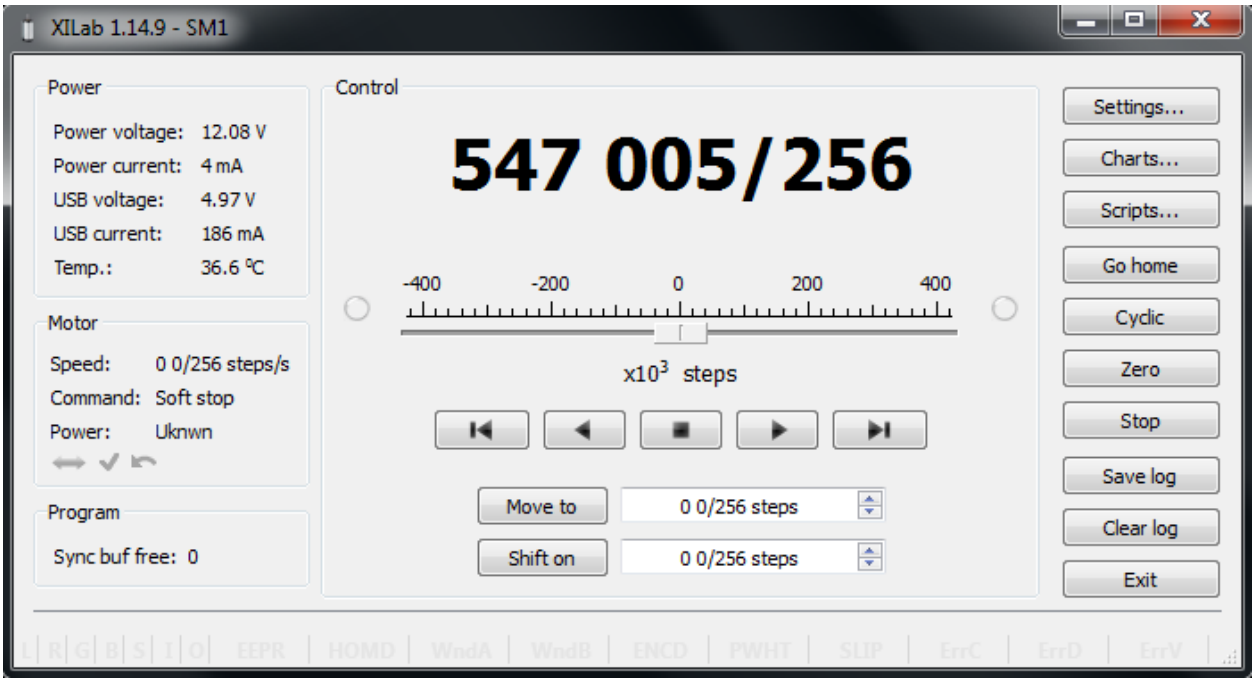
Connect the stage to the controller. Connect regulated power supply to the controller. Ground the controller or the power supply unit. Connect the controller to the computer using a USB-A - mini-USB-B cable.

LED indicator on the controller board will start to flash. New Hardware Wizard will start after the first controller is connected to the computer.



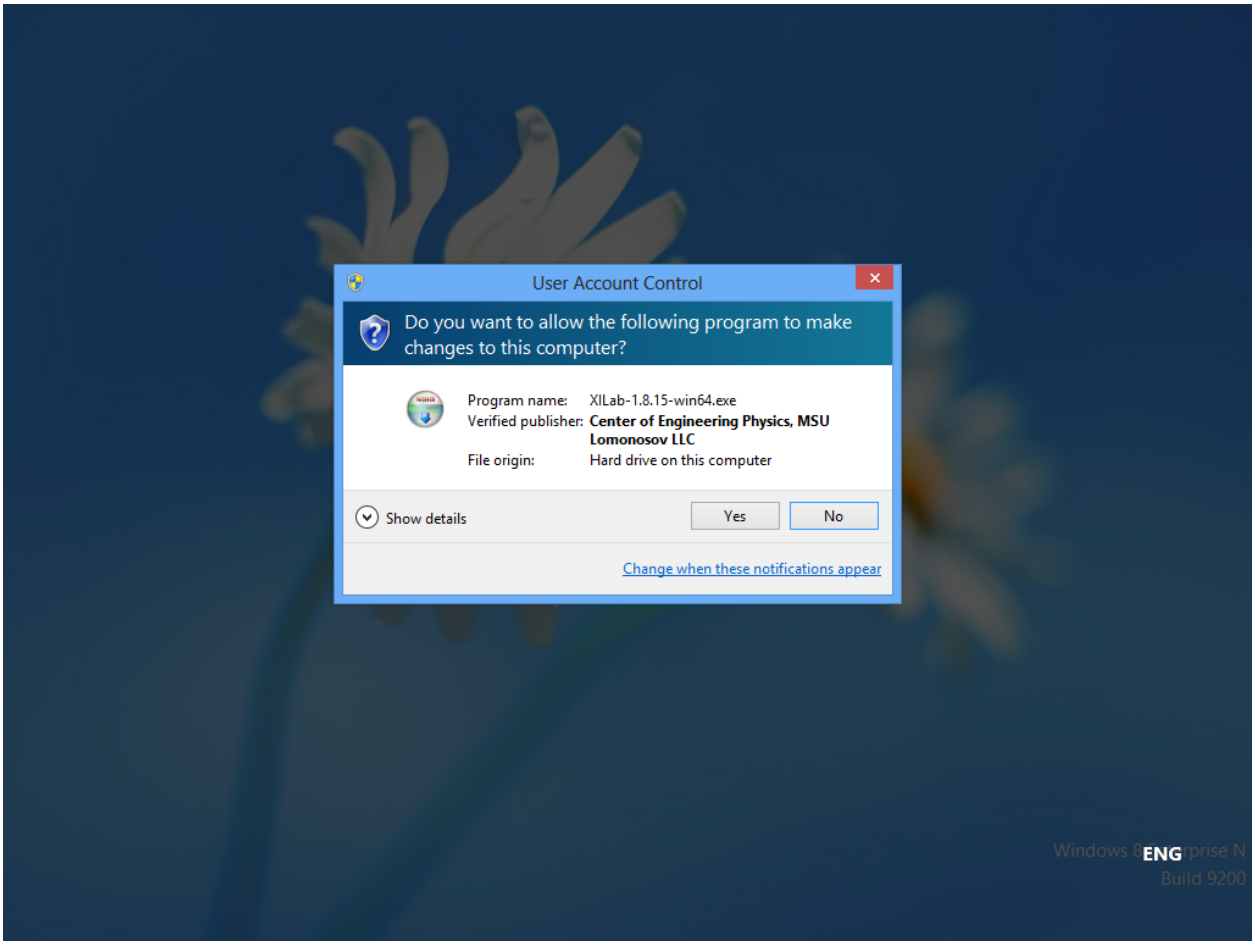
Wait until Windows detects the new device and installs the drivers for it. After the driver is successfully installed click *Rescan* or run the Xilab application again if it was closed. The system will detect the connected controller and open the main Xilab window.


XILab 1.14.9
release date: 2018-03-14
Searching devices... 1 found.
1. XIMC SM9924 on xi-com:\\.\COM25
2. Virtual XIMC, SN 1
3. Virtual XIMC, SN 2
Exit
Rescan
Open selected
Open last
Settings >>


XILab 1.14.9 - SM1
Power
Power voltage: 12.08 V
Power current: 4 mA
USB voltage: 4.97 V
USB current: 186 mA
Temp.: 36.6 °C
Motor
Speed: 0 0/256 steps/s
Command: Soft stop
Power: Uknwn
Program
Sync buf free: 0
Control
547 005/256
-400
-200
0
200
400
x10^3 steps
Move to
0 0/256 steps
Shift on
0 0/256 steps
Settings...
Charts...
Scripts...
Go home
Cyclic
Zero
Stop
Save log
Clear log
Exit

5.8.1.3 Installation on Windows 8

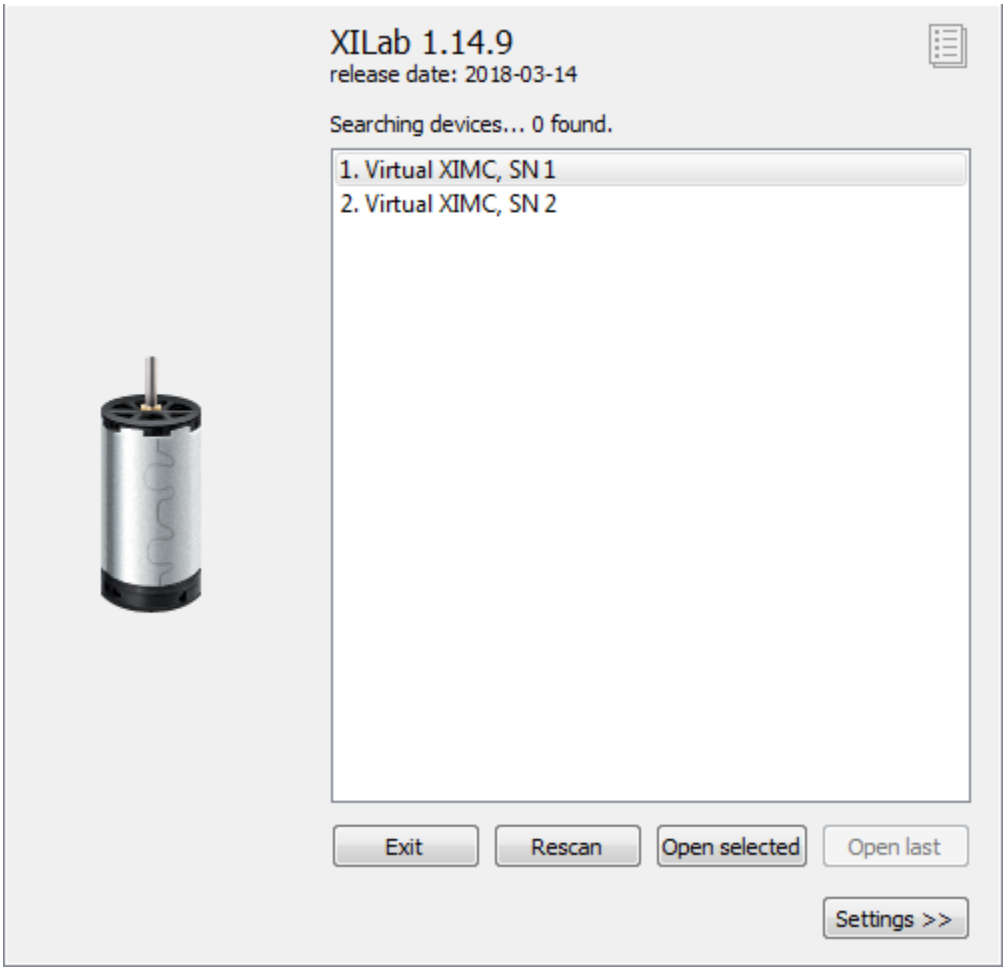
Copy the installer program file to your computer. The installer file name is "xilab-<version_name>.exe". It automatically detects whether it is running on 32-bit or 64-bit version of Windows and installs the appropriate version of XiLab. Run the installer and follow the on-screen instructions.



All the necessary software including drivers, packages and programs will be installed automatically.



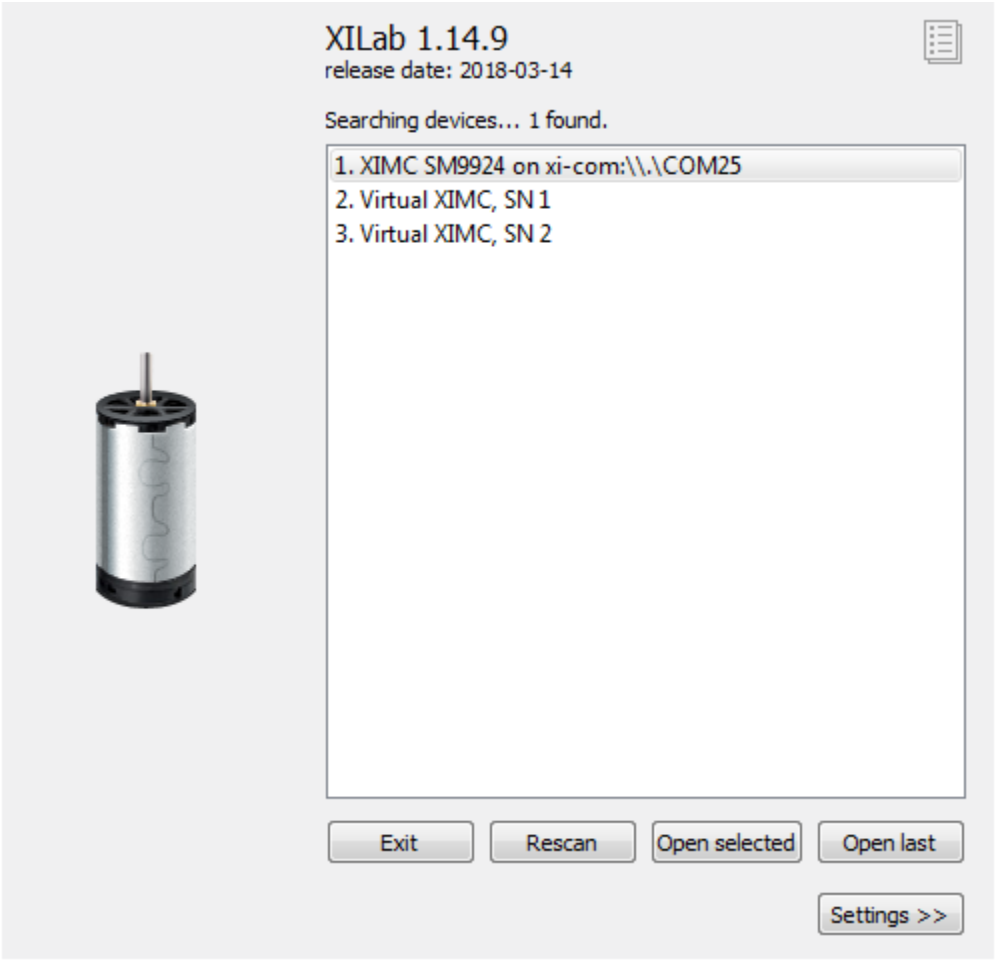
After the installation is complete the XiLab application will be started by default.

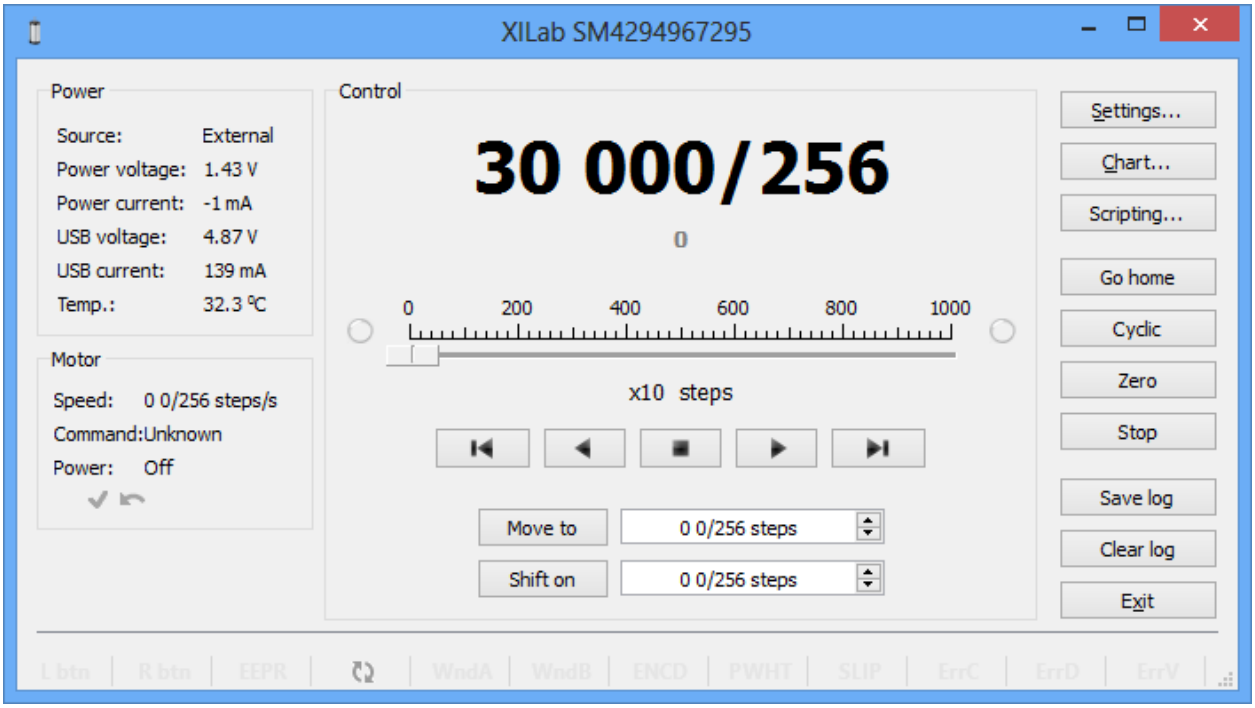


Connect the stage to the controller. Connect regulated power supply to the controller. Ground the controller or the

power supply unit. Connect the controller to the computer using a USB-A - mini-USB-B cable.

LED indicator on the controller board will start to flash. New Hardware Wizard will start after the first controller is connected to the computer. Wait until Windows detects the new device and installs the drivers for it. After the driver is successfully installed, click *Rescan* or run the Xilab application again, if it was closed. The system will detect the connected controller and open the main Xilab window.



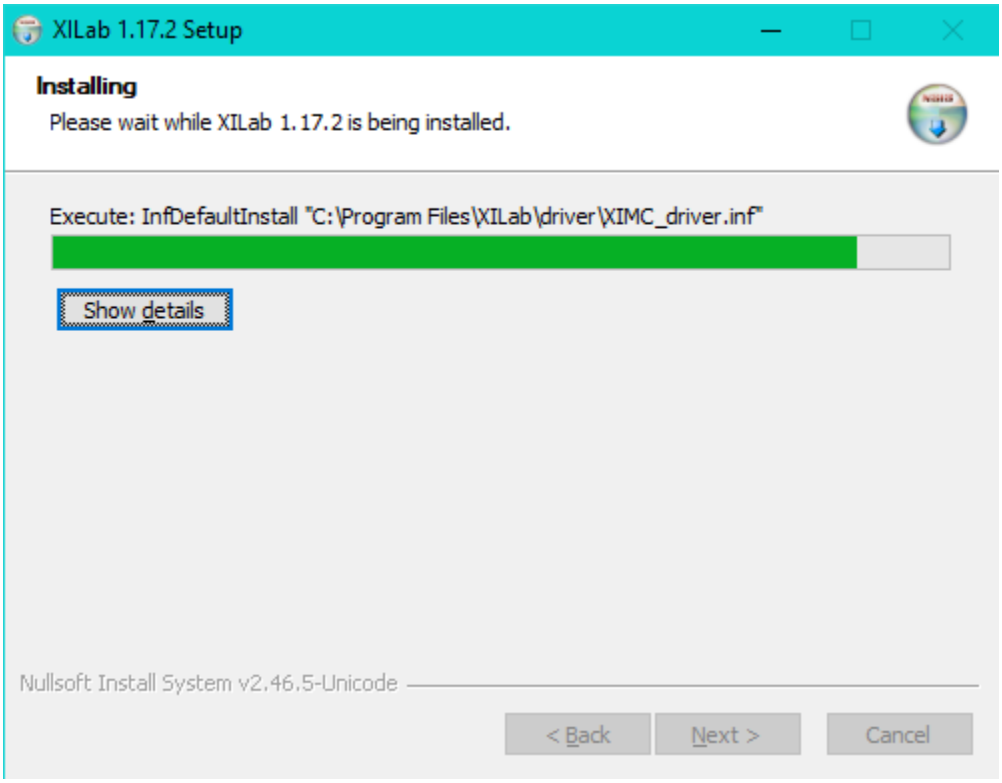


5.8.1.4 Installation on Windows 10

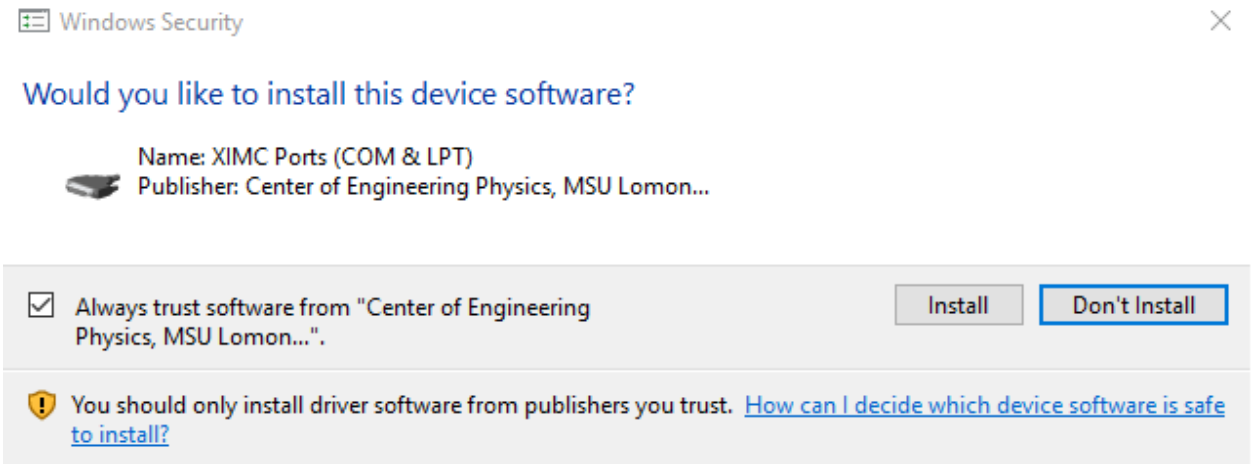
Copy the installer program file to your computer. The installer file name is "xilab-<version_name>.exe". It automatically detects whether it is running on 32-bit or 64-bit version of Windows and installs the appropriate version of XiLab.



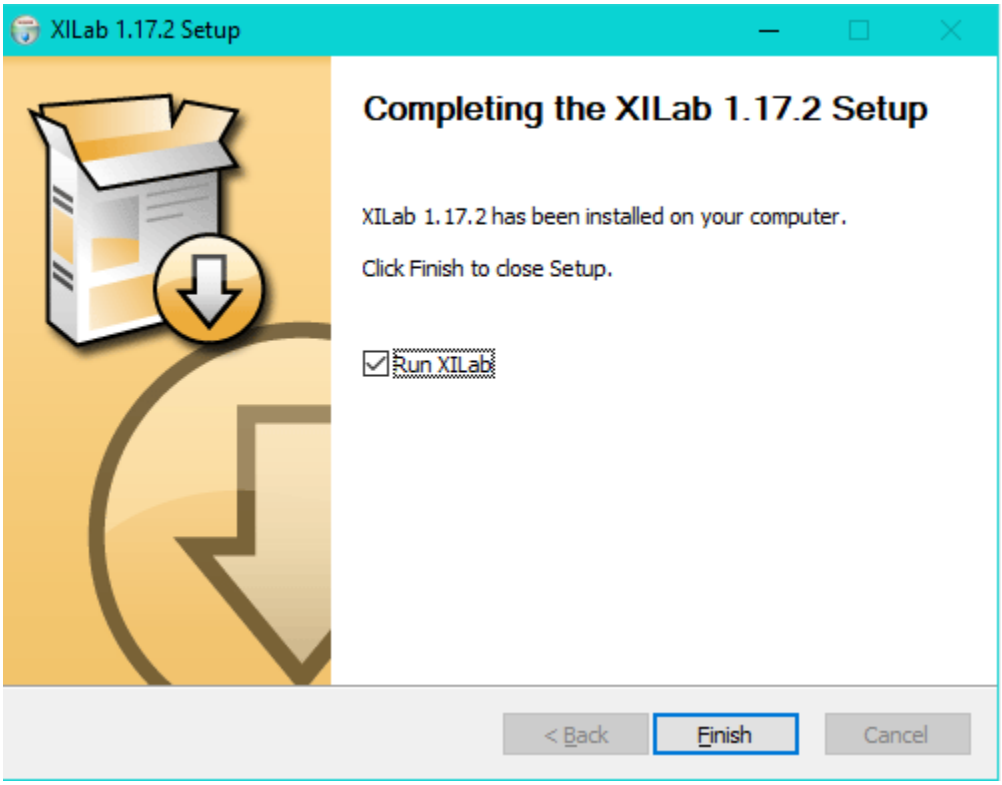
Run the installer and follow the on-screen instructions.



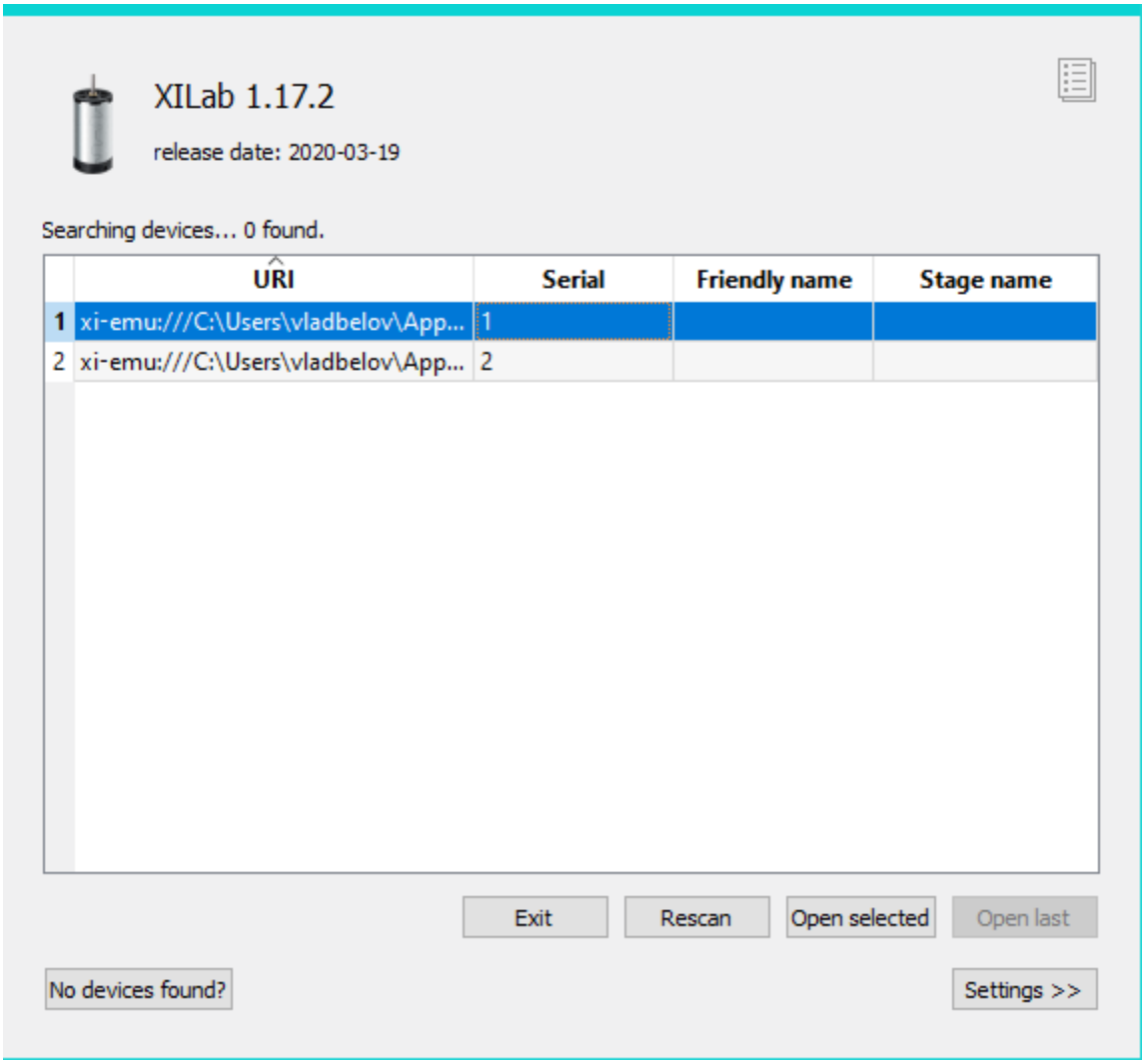
All the necessary software, packages and programs will be installed automatically.



Click *Install* to get the Ximc conroller driver.



After the installation is complete the XiLab application will be started by default.



Connect the stage to the controller. Connect regulated power supply to the controller. Ground the controller or the power supply unit. Connect the controller to the computer using a USB-A - mini-USB-B cable. LED indicator on the controller board will start to flash.

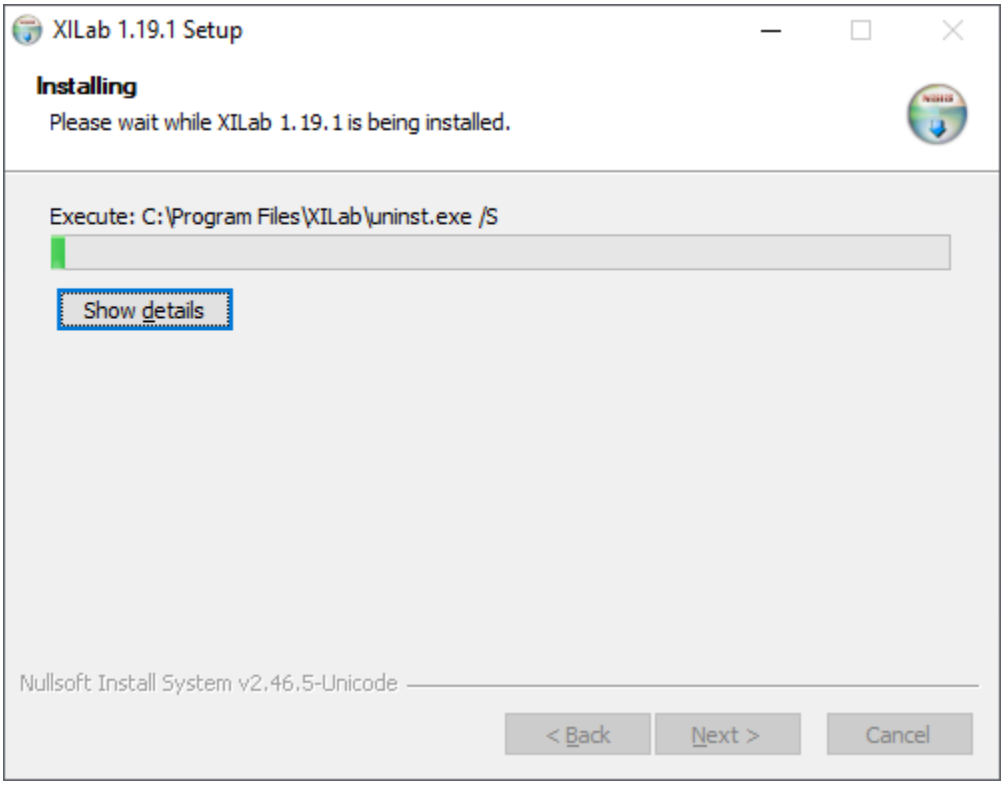
Wait until Windows detects the new device and click *Rescan* or run the Xilab application again if it was closed. The system will detect the connected controller and open the main Xilab window.

5.8.1.5 Installation on Windows 11

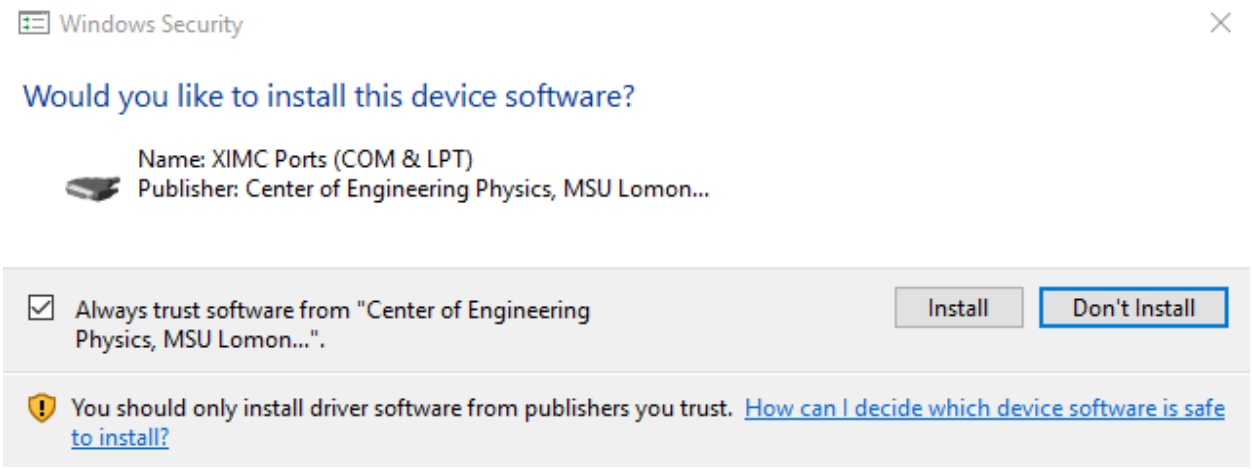
Copy the installer program file to your computer. The installer file name is "xilab-<version_name>.exe". It automatically detects whether it is running on 32-bit or 64-bit version of Windows and installs the appropriate version of XiLab.



Run the installer and follow the on-screen instructions.



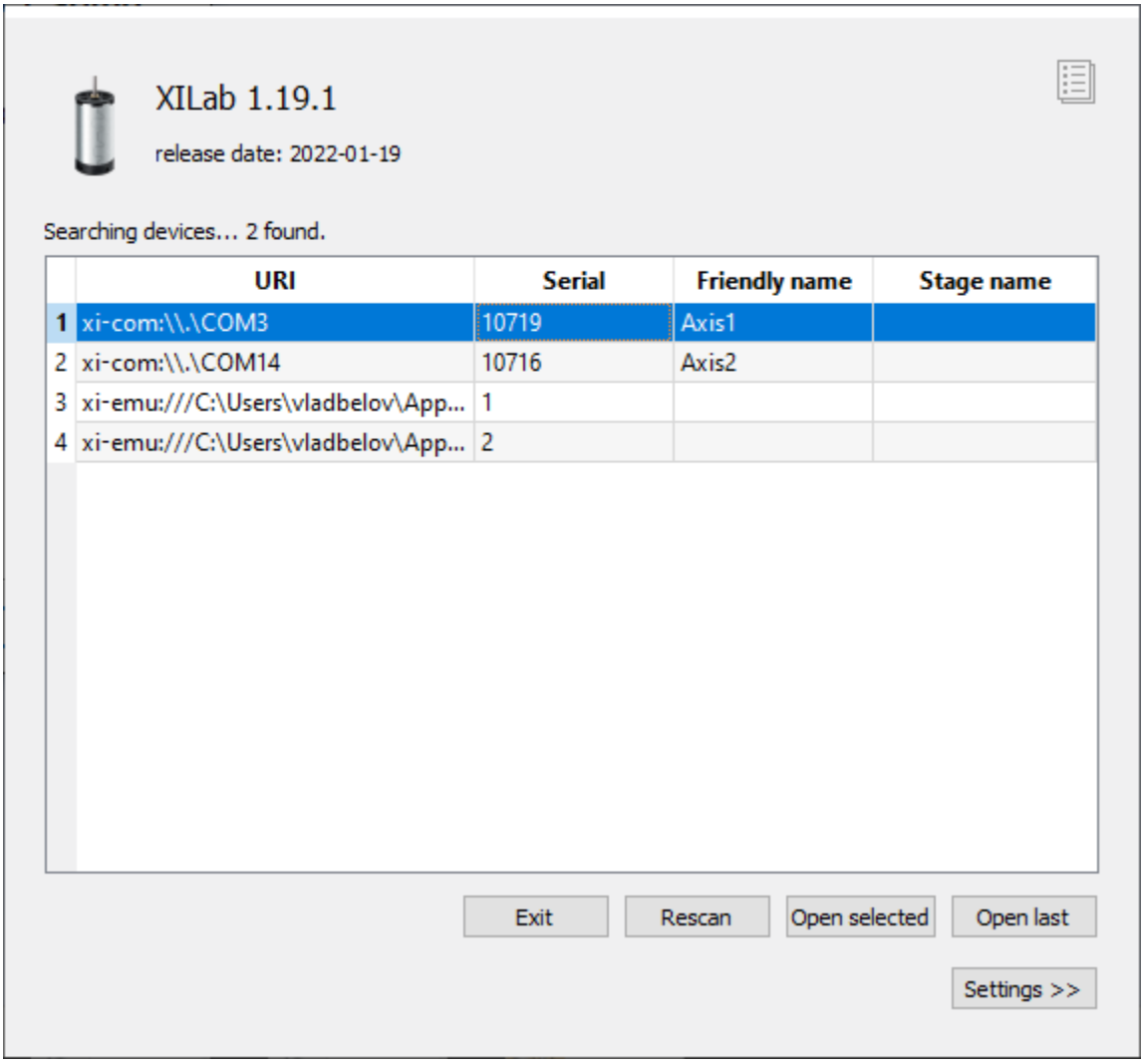
All the necessary software, packages and programs will be installed automatically.



Click *Install* to get the Ximc conroller driver.



After the installation is complete the XiLab application will be started by default.



Connect the stage to the controller. Connect regulated power supply to the controller. Ground the controller or the power supply unit. Connect the controller to the computer using a USB-A - mini-USB-B cable. LED indicator on the controller board will start to flash.

Wait until Windows detects the new device and click *Rescan* or run the Xilab application again if it was closed. The system will detect the connected controller and open the main Xilab window.

5.8.2 Installation on Linux

XILab package for Linux is distributed in AppImage format - Linux file that contains an application and everything the application needs to run (e.g., libraries, icons, fonts, translations, etc.) To run XiLab just download the application, make it executable, and run. No need to install. No system libraries or system preferences are altered.

There are two main ways to make an AppImage executable:

1. With the GUI:

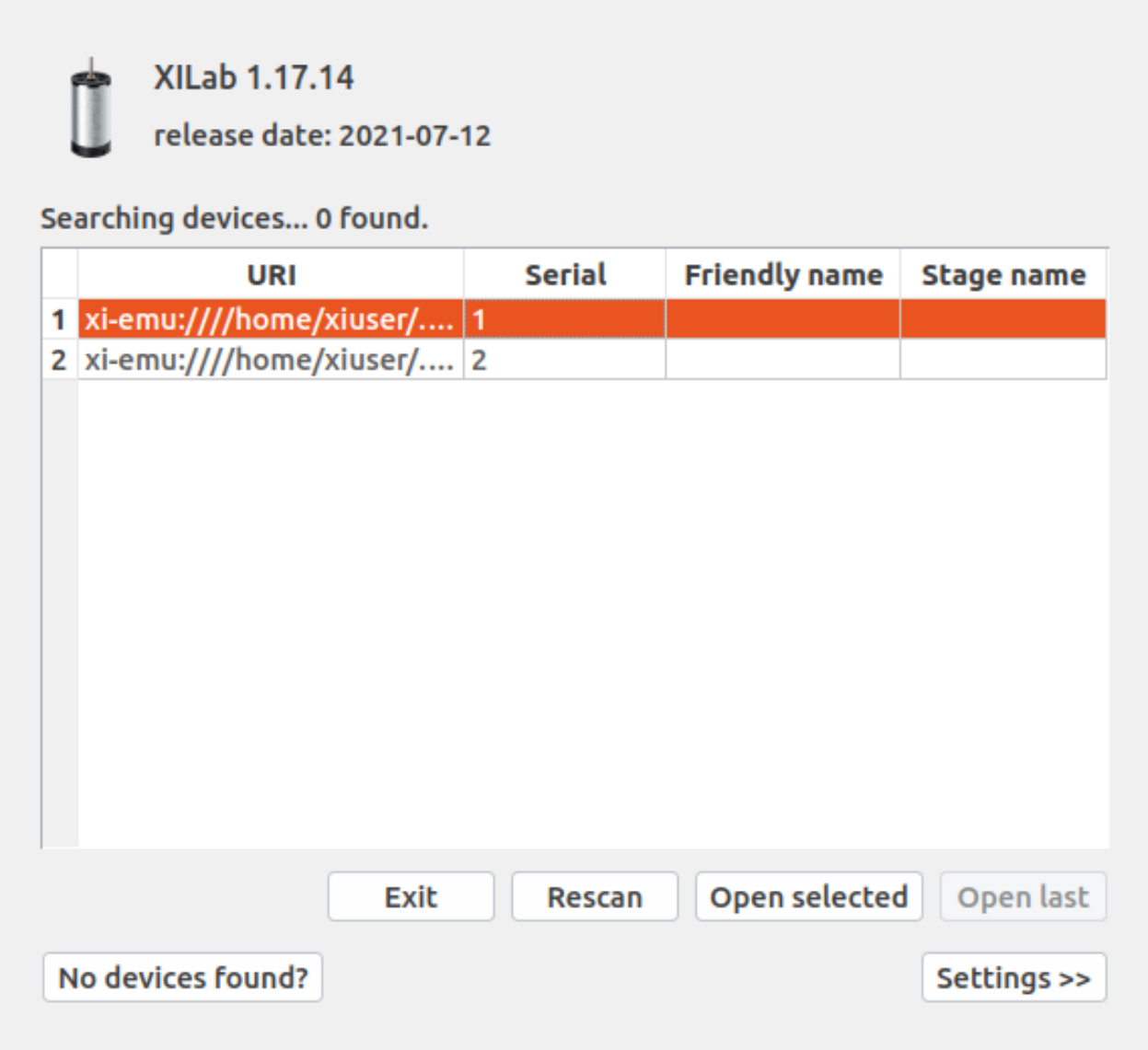
- Open your file manager and browse to the location of the AppImage;
- Right-click on the AppImage and click the “Properties” entry;
- Switch to the “Permissions” tab and click the “Allow executing file as program” checkbox if you are using a Nautilus-based file manager (Files, Nemo, Caja), or click the “Is executable” checkbox if you are using Dolphin,

or change the “Execute” drop down list to “Anyone” if you are using PCManFM;

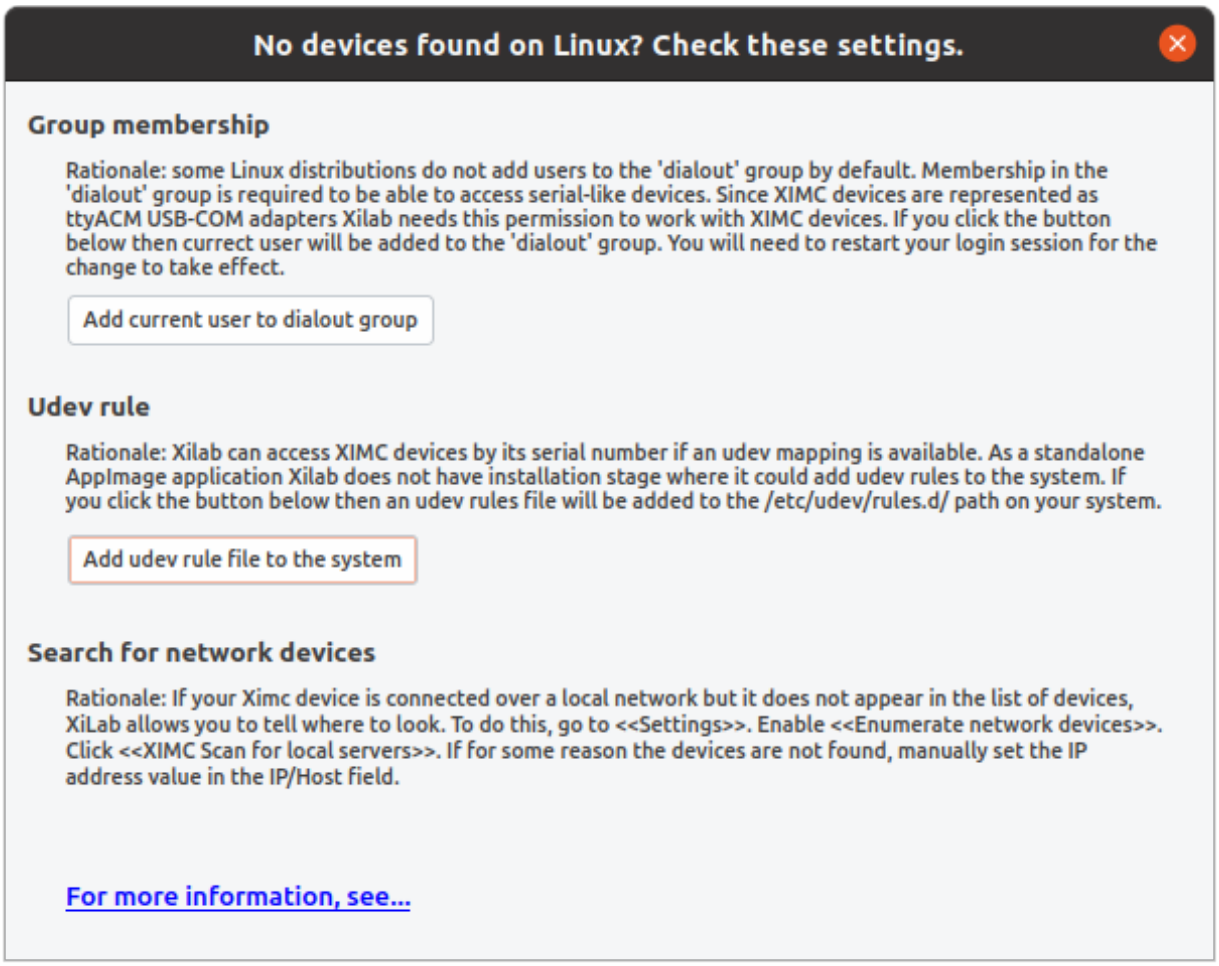
- Close the dialog;
- Double-click on the AppImage file to run.

2. On the command line:

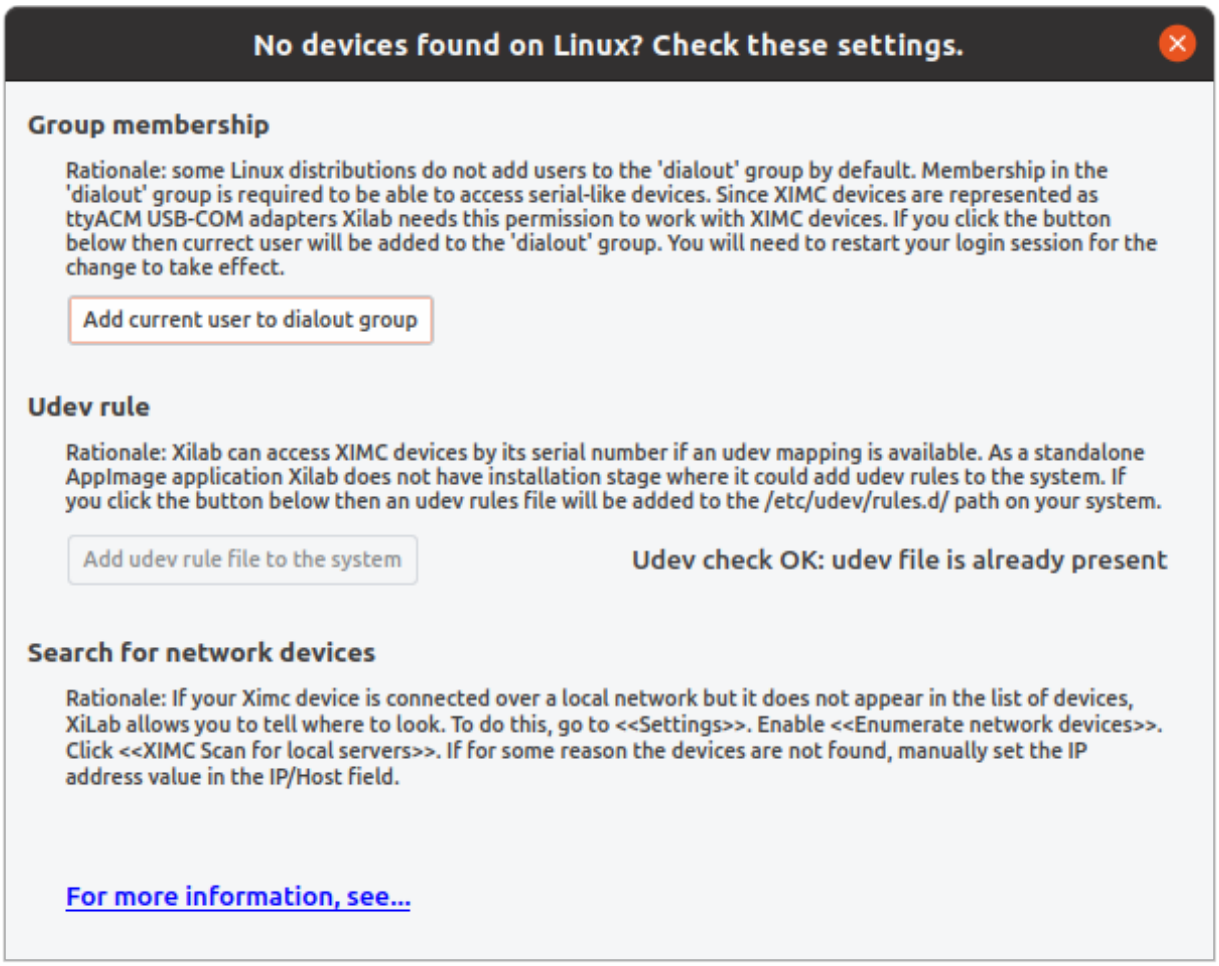
```
chmod a+x xilab-<version>-x86_64.AppImage
./xilab-<version>-x86_64.AppImage
```



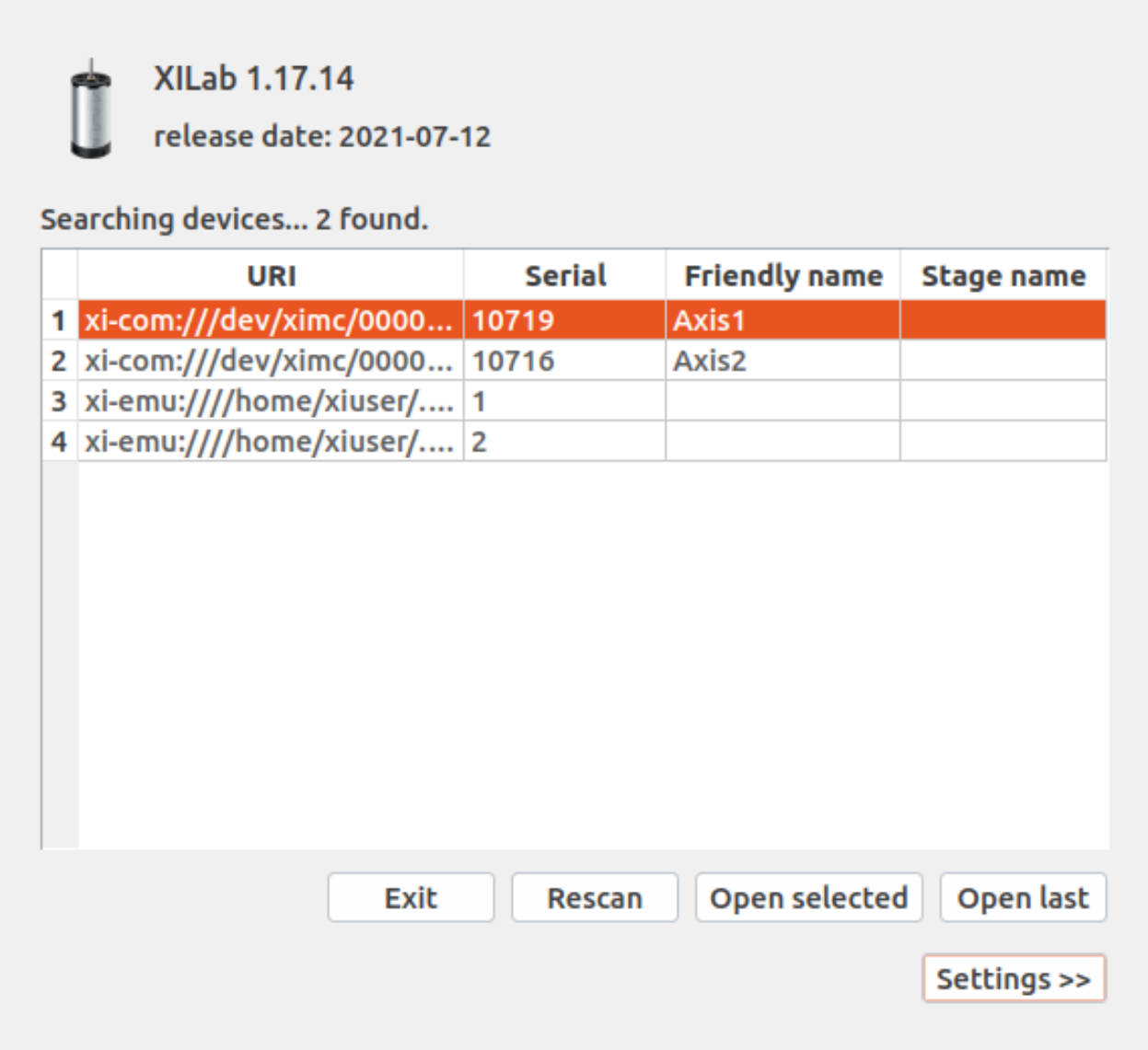
On the first run XiLab may not found the usb-connected controllers. To enumerate XIMC devices XiLab needs the available udev mapping. As a standalone AppImage application XiLab doesn’t have installation stage which can add the udev rules to the system. Click the *No devices found?* button on the XiLab start window then click *Add udev rule file to the system*.



Some Linux distributions do not add users to the “dialout” group by default. Membership in the “dialout” group is required to be able to access serial-like devices. Since XIMC devices are represented as ttyACM USB-COM adapters XiLab needs this permission to work with XIMC devices. Click *Add current user to the dialout group* button and restert your login session for the chage to take effect.



Important: Xilab application requires X-server (graphic mode) for operation.



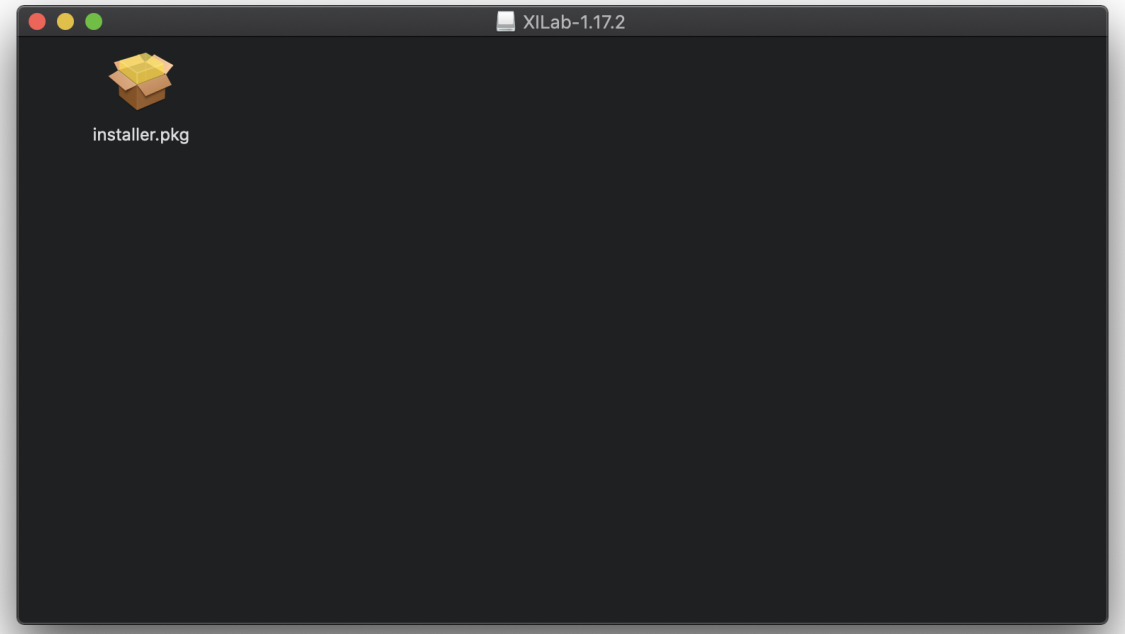
5.8.3 Installation on MacOS



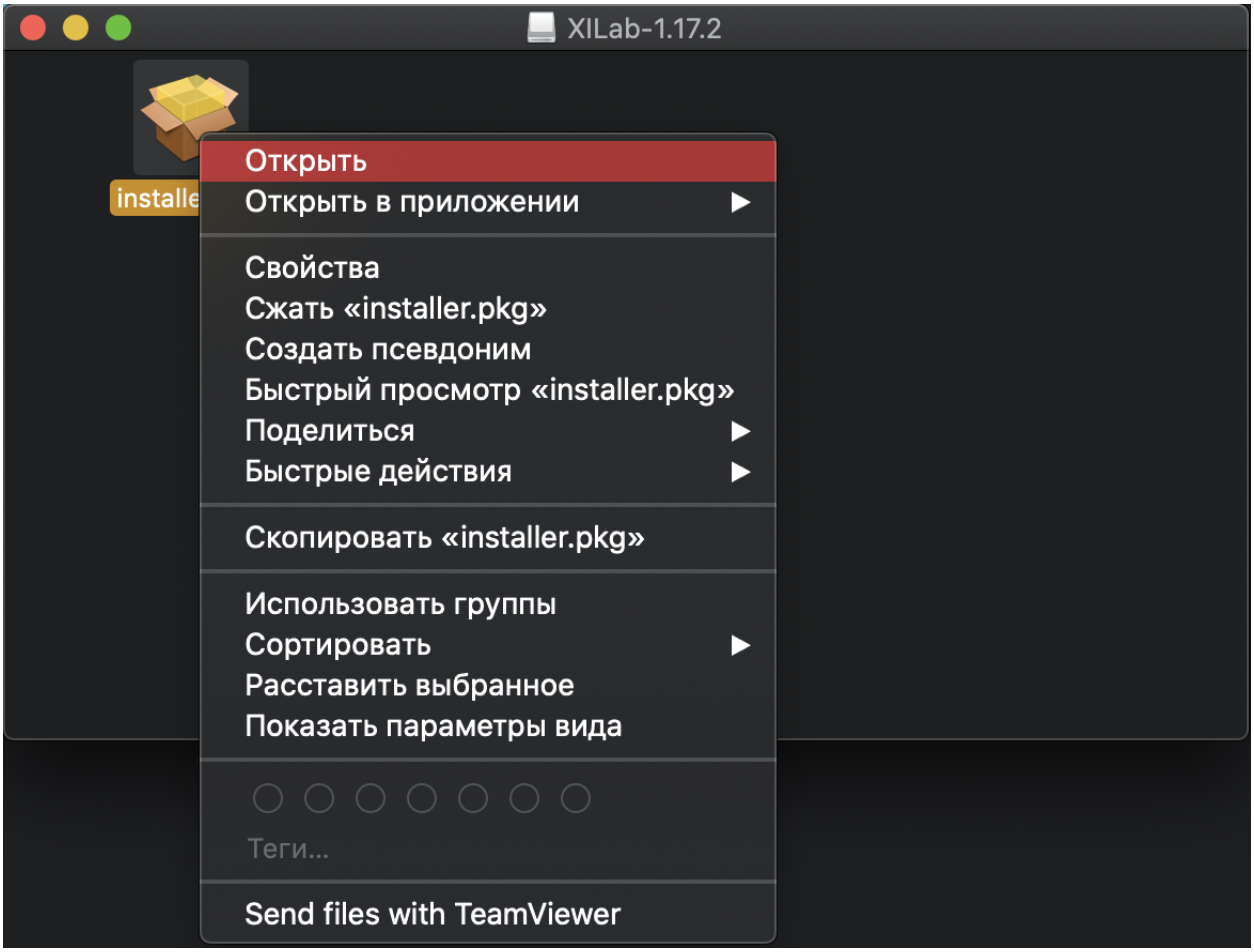
Copy the file with the installer archive to your computer. The archive with the installation program is named "xi-lab–osx64.tar.gz".



Unpack the archive by a mouse click.



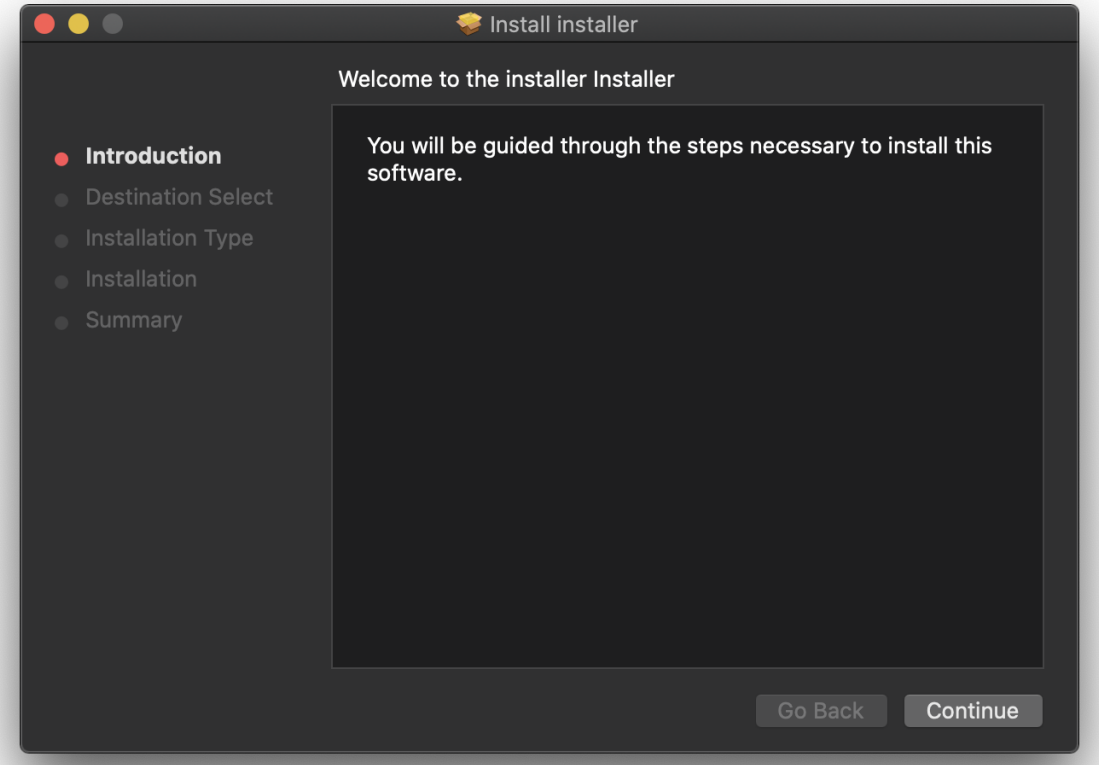
Make right button click on installer.pkg.


XILab-1.17.2
installe
Открыть
Открыть в приложении
Свойства
Сжать «installer.pkg»
Создать псевдоним
Быстрый просмотр «installer.pkg»
Поделиться
Быстрые действия
Скопировать «installer.pkg»
Использовать группы
Сортировать
Расставить выбранное
Показать параметры вида
Теги...
Send files with TeamViewer

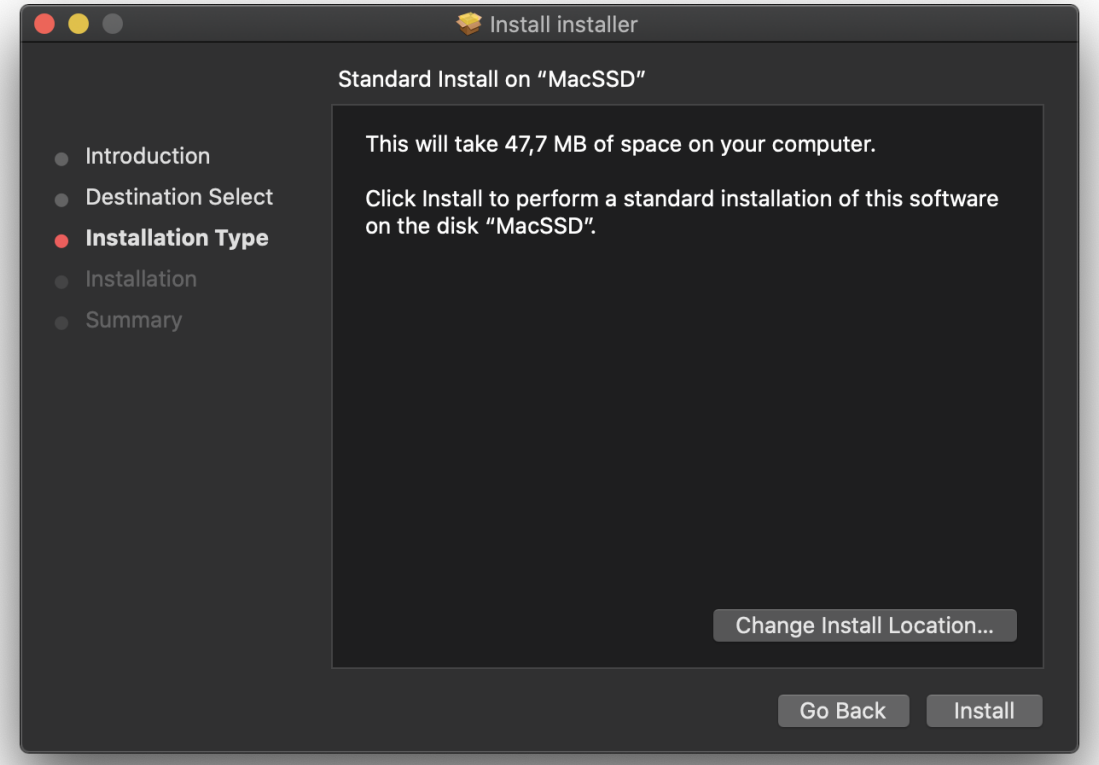
Choose “Open”.


macOS cannot verify the developer of “installer.pkg”. Are you sure you want to open it?
By opening this app, you will be overriding system security which can expose your computer and personal information to malware that may harm your Mac or compromise your privacy.
This item is on the disk image “xilab-1.17.2.dmg”. Chrome downloaded this disk image today at 18:25 from files.xisupport.com.
Open
Cancel

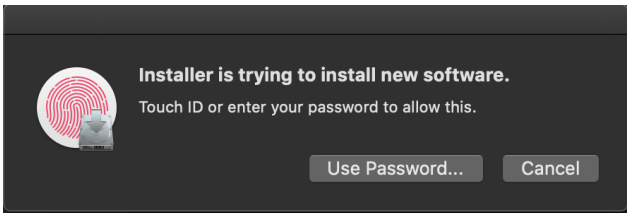
Choose “Open”.



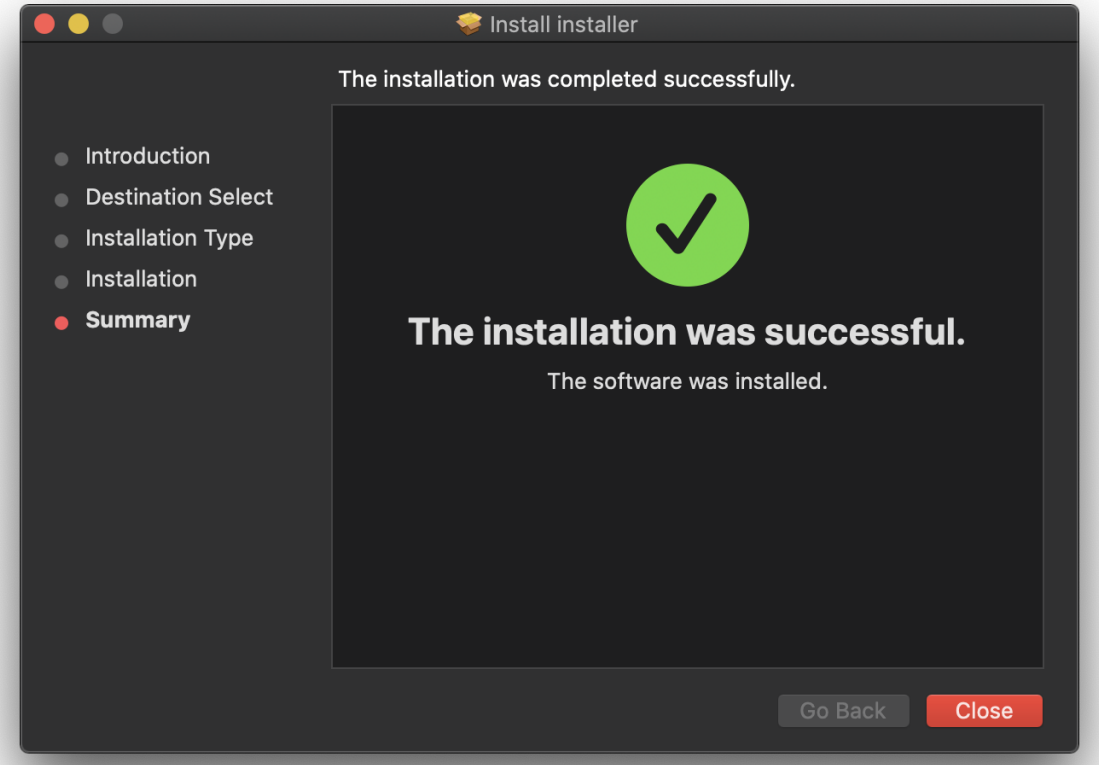
Select “Continue” in the main window of the installer.



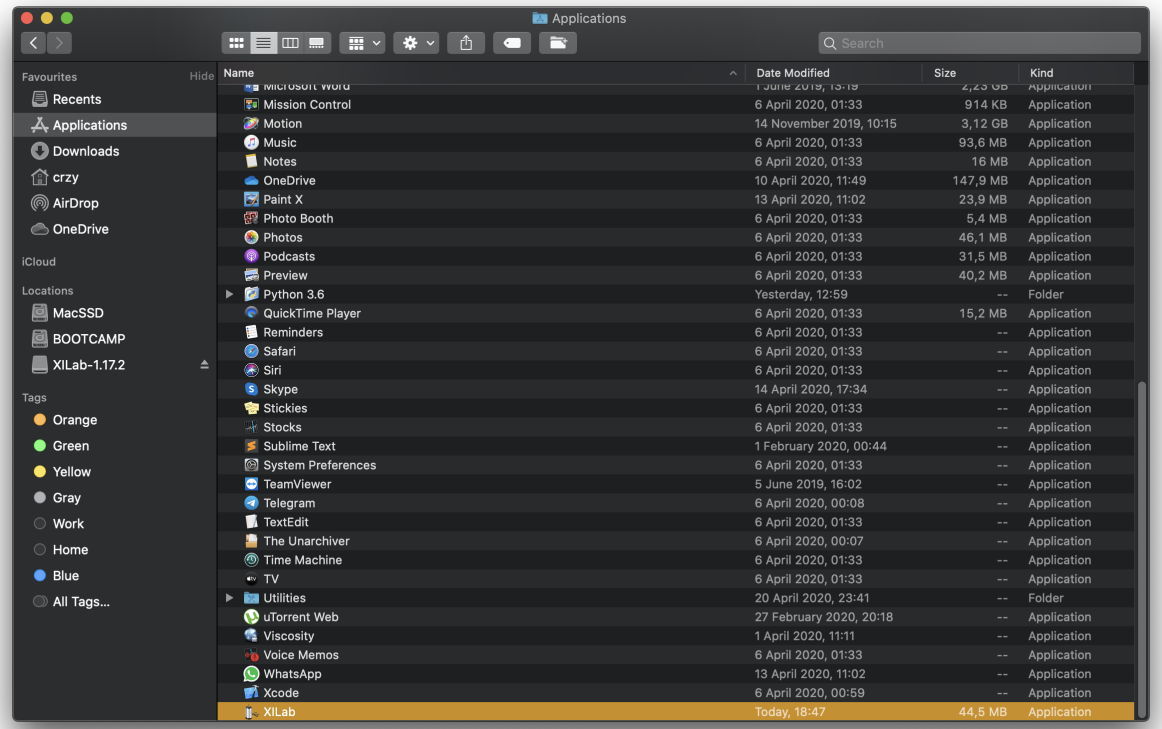
Now select "Install."



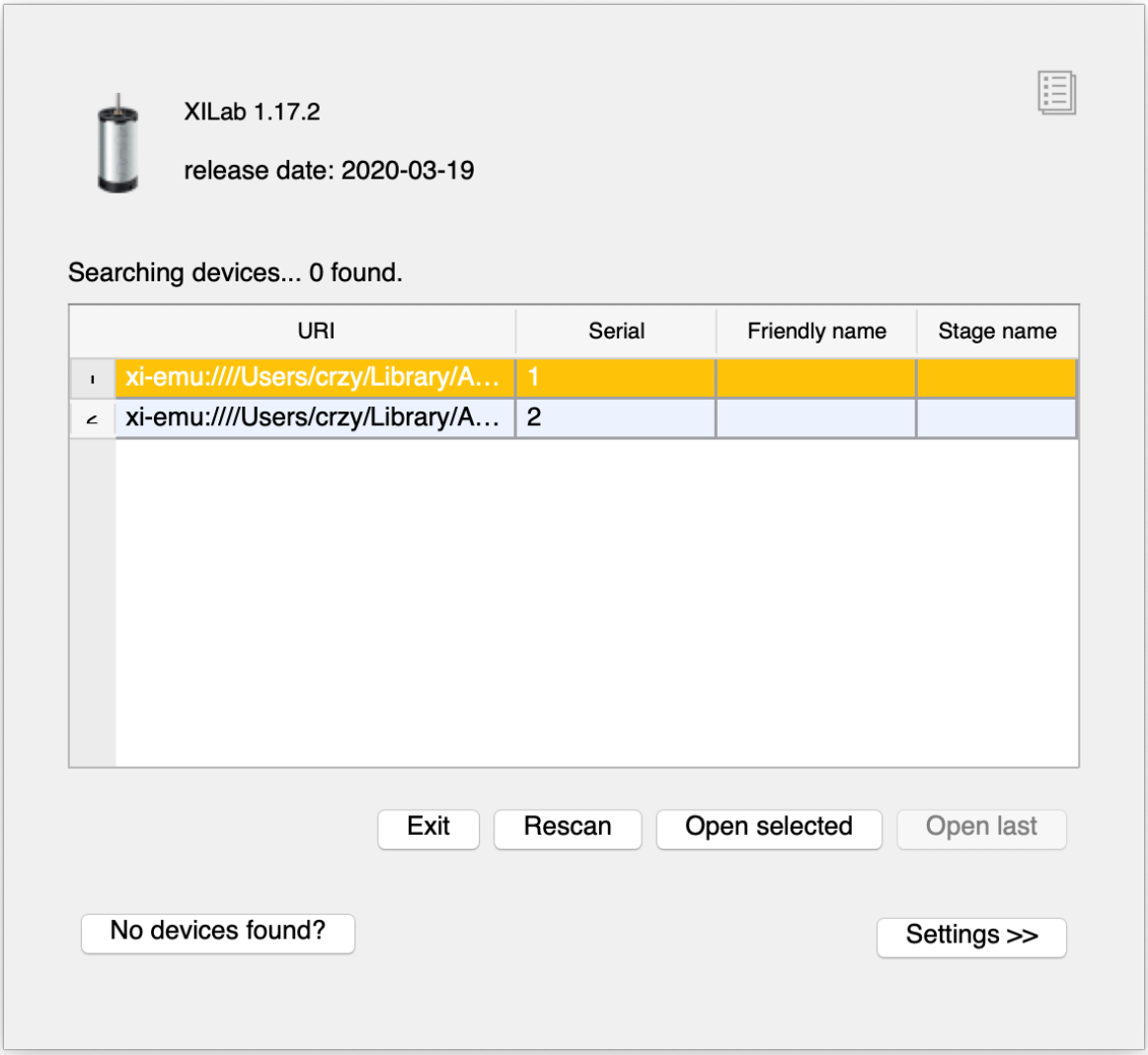
Enter the password.



Wait until the installation is complete.



Select the XILab application in the Programs block


XILab 1.17.2
release date: 2020-03-19
Searching devices... 0 found.
URI
Serial
Friendly name
Stage name
xi-emu:////Users/crzy/Library/A…
1
xi-emu:////Users/crzy/Library/A…
2
Exit
Rescan
Open selected
Open last
No devices found?
Settings >>

Start it.

CHAPTER

SIX

PROGRAMMING

6.1 Programming guide

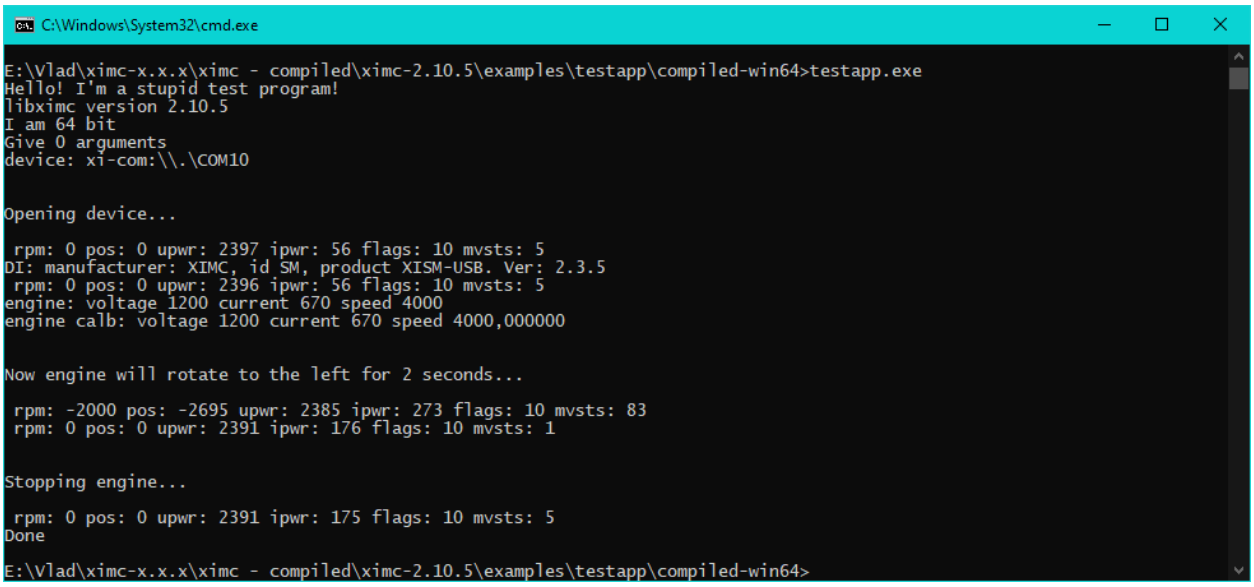
6.1.1 Working with controller in Visual Studio

Download VisualStudio example from the Software page.

Note: Testapp can be built using *testapp.sln*. Library must be compiled with MS Visual C++ too. Make sure that Microsoft Visual C++ Redistributable Package 2013 is installed.

Open solution *examples/testapp/testapp.sln*, build and run from the IDE.

Extract the archive and run *"testapp"* program.



The command prompt opens. You will see a message: "Hello! I'm a stupid test program!"

The program reports the version of the library used, as well as its bit depth. Also *"testapp"* program indicates which port it holds.

Note: **set_bindy_key("keyfile.sqlite");** Must be called before any call to *"enumerate_devices"* or *"open_device"* if you wish to use network-attached controllers. Accepts both absolute and relative paths, relative paths are resolved

relative to the process working directory. If you do not need network devices then “set_bindy_key” is optional.

After opening the device, the program reads the data fields from the *“status_t”* structure reference.

rpm	int CurSpeed	Motor shaft speed
pos	float CurPosition	Current position
upwr	int Upwr	Power supply voltage, tens of mV
ipwr	int Ipwr	Engine current
flags	unsigned int Flags	Status flags
mvsts	unsigned int MvCmdSts	Move command state

*Function result_t XIMC_API get_device_information (device_t id, device_information_t *device information)* - Return device information

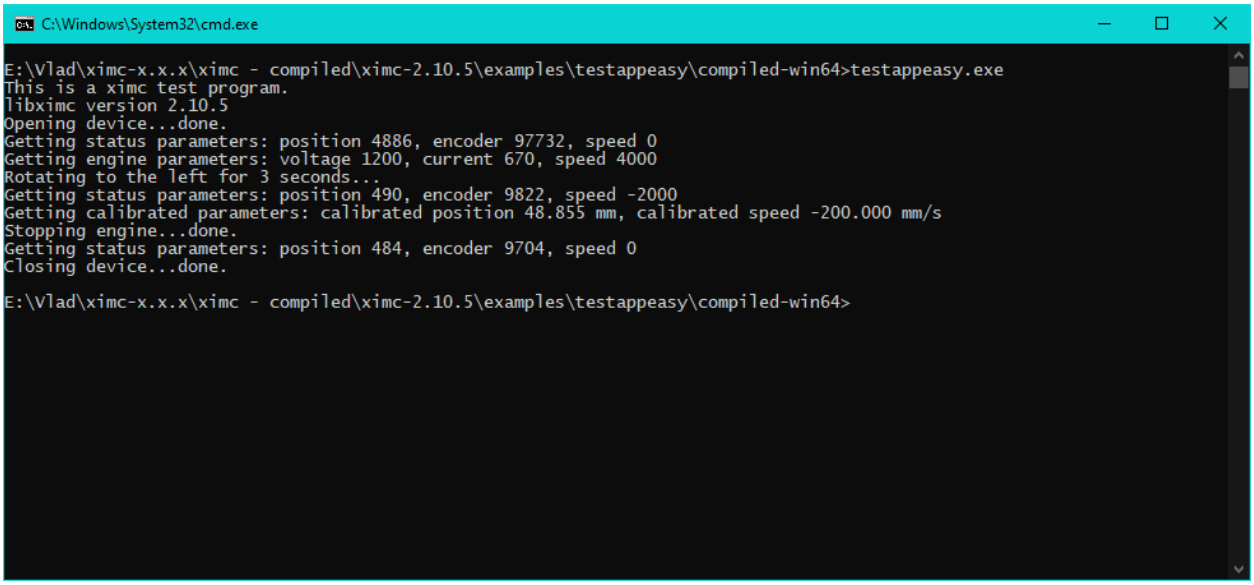
*Function result_t XIMC_API get_engine_settings (device_t id, engine_settings_t *engine settings)* - Read engine settings

engine_settings_calb_t Struct Reference - result_t XIMC_API set_engine_settings_calb (device_t id, const engine_settings_calb_t *engine_settings - calb, const calibration_t *calibration)

After, the *testapp* program executes the command *“command_left”* for 2 seconds. The *“command_left”* command is successfully executed, the *“command_stop”* command is called.

Important: At the end of the program, the command **“close_device”** must be called.

The *“testappeasy”* program isn‘t so much different from the *“testapp”* program. Open solution *examples/testappeasy/testappeasy.sln*, build and run from the IDE.



The command prompt opens. You will see a message: “This is a ximc test program.”

The program reports the version of the library used.

Note: **set_bindy_key(“keyfile.sqlite”);** Must be called before any call to *“enumerate_devices”* or *“open_device”*

if you wish to use network-attached controllers. Accepts both absolute and relative paths, relative paths are resolved relative to the process working directory. If you do not need network devices then "set_bindy_key" is optional.

Using the *"open_device"* command, *"testappeasy"* program opens the device in exclusive access mode.

Warning: Libximc library opens the controller in exclusive access mode. Any controller opened with libximc (XiLab also uses this library) needs to be closed before it may be used by another process.

After, the *testappeasy* program executes the command *"command_left"* for 3 seconds. The *"command_left"* command is successfully executed, the *"command_stop"* command is called.

The comand *"calibration.A = 0.1;"* - Setting calibration constant to 0.1 (one controller step equals this many units)

The command *"calibration.MicrostepMode = engine_settings.MicrostepMode;"* - To set microstep mode to convert microsteps to calibrated units correctly.

After the *"testappeasy"* program reads calibrated device status from a device.

At the end, a *"command_stop"* command is sent to the device. The *"close_device"* - closes the specified device.

6.1.2 Working with controller in Delphi

Download Delphi example from the Software page.

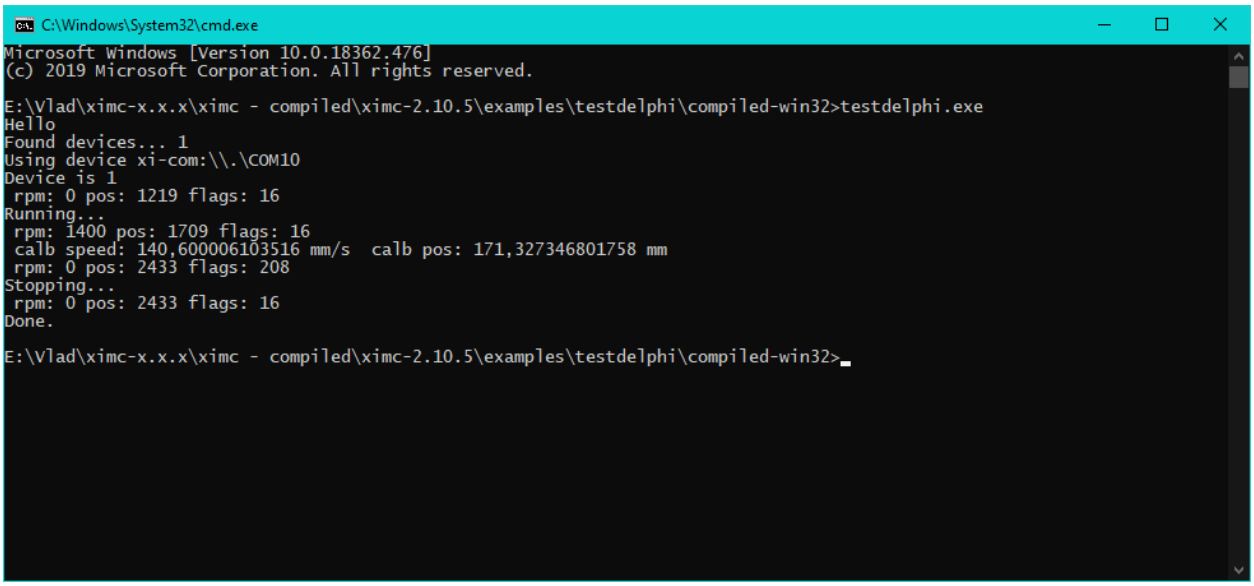
Note: Wrapper for libximc.dll is a unit wrappers/delphi/ximc.pas Console test application for is it located at *"testdelphi"*. **Tested with Delphi 6 and only 32-bit version**. Just compile, place DLL near the executable and run the program.

Warning: To work this example copy files ximc.pas, keyfile.sqlite, libximc.dll, xiwrapper.dll, bindy.dll from ximc folder next to *"testdelphi.dpr"* and run *"testdelphi.dpr"* with your Delphi IDE.

To run compiled example without IDE (through the command line), copy all files (keyfile.sqlite, libximc.dll, ... etc) in "compiled-win32" directory and run *"testdelphi.exe"*

Open solution *examples/testdelphi/testdelphi.dpr*, build and run from the IDE.

Extract the archive and run *"testdelphi.exe"* program.



The command prompt opens. You will see a message: “Hello”

The demo program *“testdelphi”* uses the command **“result_t XIMC_API get_status (device_t id, status_t ,status)”** to output the current state of the device to the command prompt. Also *“testdelphi”* program indicates which port it holds.

Note: set_bindy_key(“keyfile.sqlite”); Must be called before any call to *“enumerate_devices”* or *“open_device”* if you wish to use network-attached controllers. Accepts both absolute and relative paths, relative paths are resolved relative to the process working directory. If you do not need network devices then “set_bindy_key” is optional.

Using the *“open_device”* command, *“testdelphi”* program opens the device in exclusive access mode.

> **Warning:** Libximc library opens the controller in exclusive access mode. Any controller opened with libximc (XiLab also uses this library) needs to be closed before it may be used by another process.

After opening the device, the program reads the data fields from the *“status_t”* structure reference.

rpm	int CurSpeed	Motor shaft speed
pos	float CurPosition	Current position
flags	unsigned int Flags	Status flags

*Function result_t XIMC_API get_engine_settings (device_t id, engine_settings_t *engine settings)* Read engine settings

After, the *“testdelphi”* program executes the command *“command_right”*. The *“command_right”* command is successfully executed, the *“command_stop”* command is called.

The comand “calibration.A = 0.1;” - Setting calibration constant to 0.1 (one controller step equals this many units)

The command “calibration.MicrostepMode = engine_settings.MicrostepMode;” - To set microstep mode to convert microsteps to calibrated units correctly.

After the “testdelphi” program reads calibrated device status from a device.

Warning: At the end of the program, the command **"close_device"** must be called.

6.1.3 Working with controller in LabView

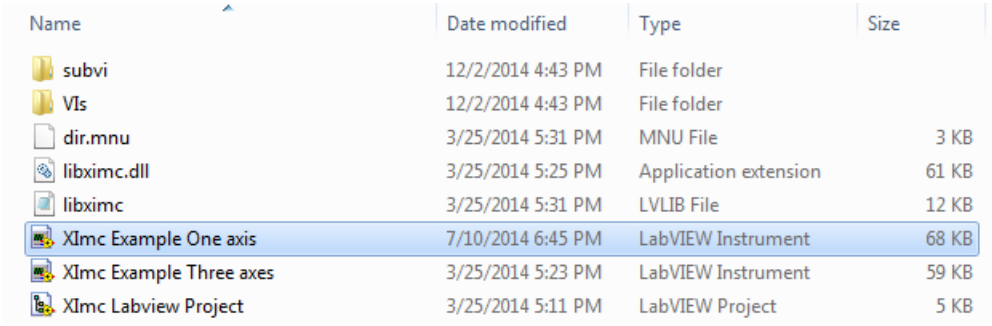
Download Labview example from the Software page.

Warning: Our LabView examples only work for Windows. The bitness of the LabView examples should match the bitness of the operating system, **not the bitness of LabView**

Important: The examples below are for LabVIEW version 12. The correct operation of the examples with later versions of LabVIEW is not guaranteed.

For older versions of the examples, please contact technical support send an e-mail: **8smc4@standa.lt**

Extract the archive and run "Ximc Example One axis" file using Labview.



Warning: The "One axis example.vi" is a simple to use but quite complicated pseudo-xilab example that is not recommended to modify by yourself. Take a look at the "ximc simple example.vi". This example is easy to adaptate for your purposes.

LabView environment will start. You will see graphical user interface of the front panel of the example, which looks like a simplified *XILab* interface.

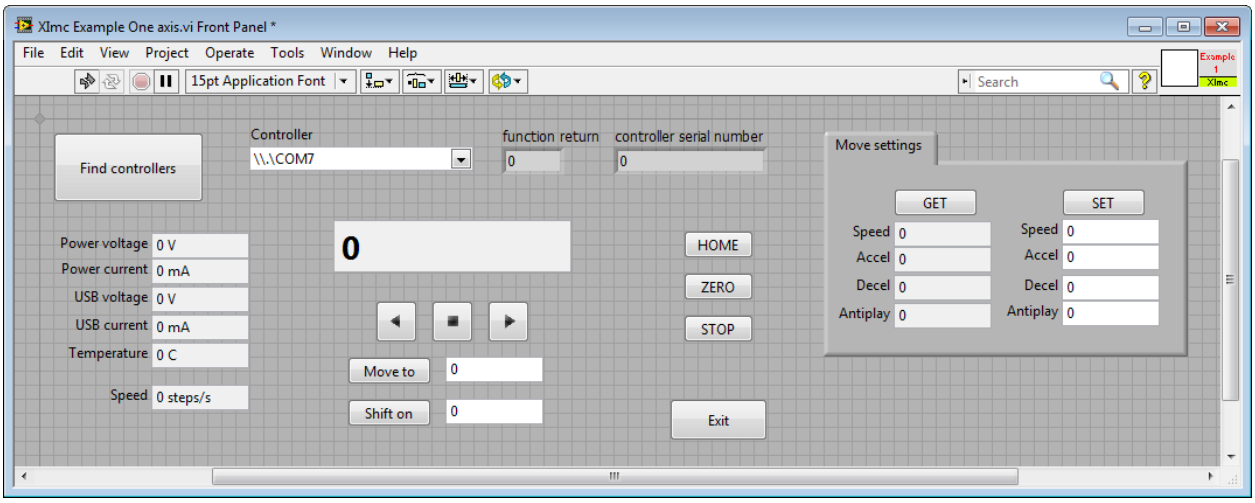
Note: LabVIEW works independently of the XiLab. XiLab is an alternative tool for interacting with the controller. It is convenient to use for device configuration and tests. All functions available through XiLab can be independently implemented using LabVIEW.

In the left part of the window you can find a "Find controllers" button to rescan available controllers, a field to pick the controller by its serial port name and an information block which displays current opened controller state (power and usb voltage and current, temperature and movement speed).

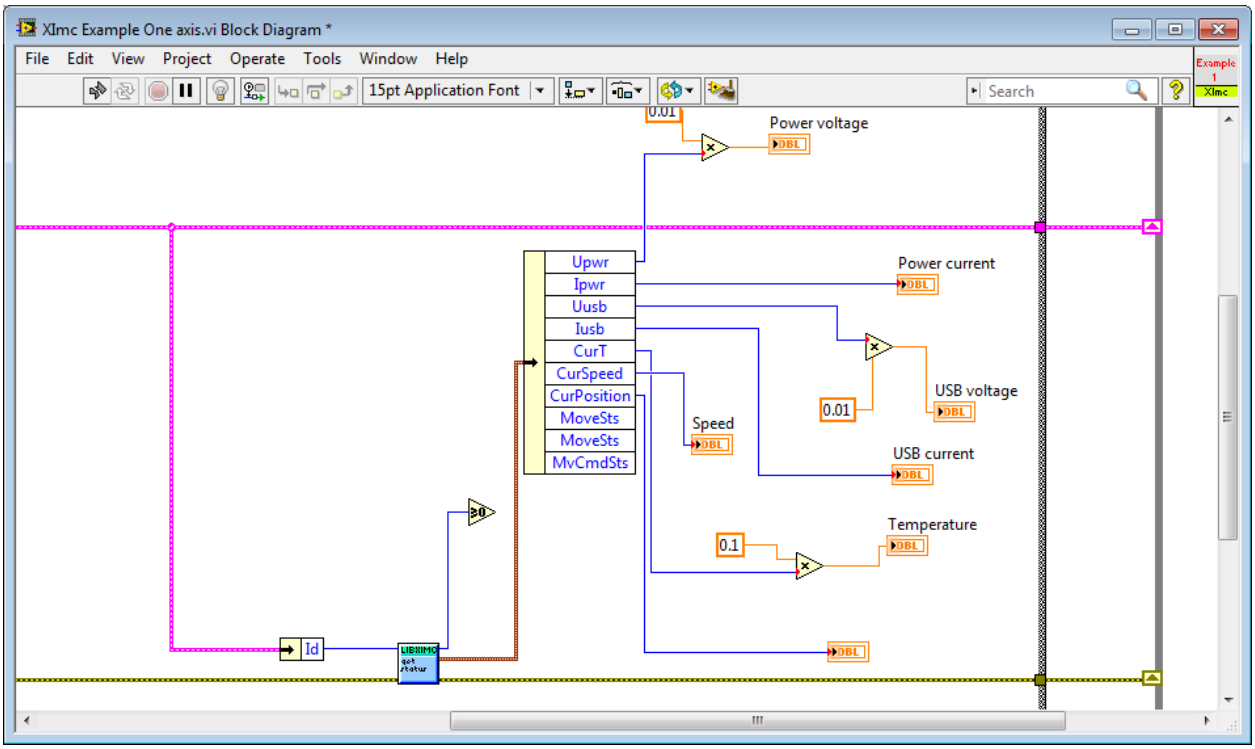
In the central part of the window you will find indication and control block. This block contains a numeric field with current position, left, right and soft stop buttons, controls to move to specified coordinate and shift on specified offset.

To the right of this block you can find "HOME", "ZERO", "STOP" and "Exit" buttons, which perform homing, zero current controller position, perform fast stop and exit the example, respectively.

The rightmost block is a “Move settings” dialog, which demonstrates how you can load and save settings to the controller. When you press its “GET” button current movement settings are loaded into the fields below this button and the “SET” button sends values from its edit fields to the controller.



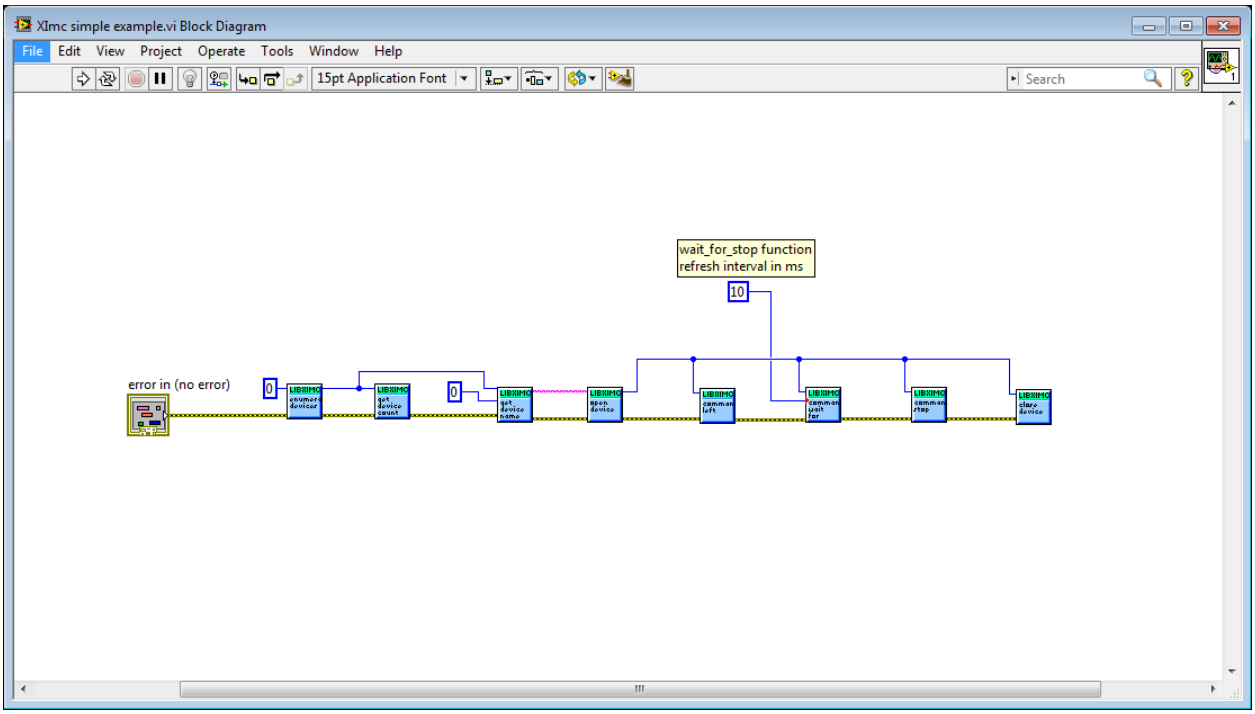
Example source can be viewed by entering edit mode. The example queries the controller status in an infinite loop and outputs the results. If you press any button on the front panel, the corresponding function block of the example will be launched.



LabVIEW uses the libximc library to interact with controllers. The interface is accessible via LabVIEW is completely analogous to the libximc library interface. A detailed description of the functions, data structures and flags can be downloaded from this link. Each function has a corresponding subVI module, which has inputs and outputs corresponding to input and output parameters of this function. To call any libximc function one should first enumerate

devices by using “device_enumerate”, then pick any device from the list and open it using “open_device” function, then pass resulting handle to desired libximc function. After you’re finished using the device it should be closed with “close_device” function.

Let’s review how to create a simple Labview program with libximc library using “Ximc simple example.vi”.



The program starts by calling enumerate_devices function and passing it the enumerate flags parameter (for details see *Programming guide*). Result of the enumerate_devices function call is an opaque pointer and is passed to the get_device_name function together with device number, whose name we want to find out (one can get total number of found devices by passing the same opaque pointer to the get_device_count function). Result of the get_device_name function call is a string, which is passed to the open_device function. This function call sequence is not mandatory (though recommended) - one can manually form device name string and pass it to the open_device function directly. Result of the open_device function call is a device handle, or a ximc.h header constant device_undefined, which is returned if libximc could not open specified device. Device handle is passed to all functions which read data from the controller, write data to the controller or send commands to the controller, together with appropriate parameters if necessary by function prototype. “Ximc simple example.vi” calls command_left, command_wait_for_stop and command_stop as an example. After you stop using the controller you need to close its handle by passing it to the close_device function. After you stop using enumerate_devices result you need to free allocated memory pointed to by this opaque pointer by passing it to the free_enumerate_devices function (not shown in this example).

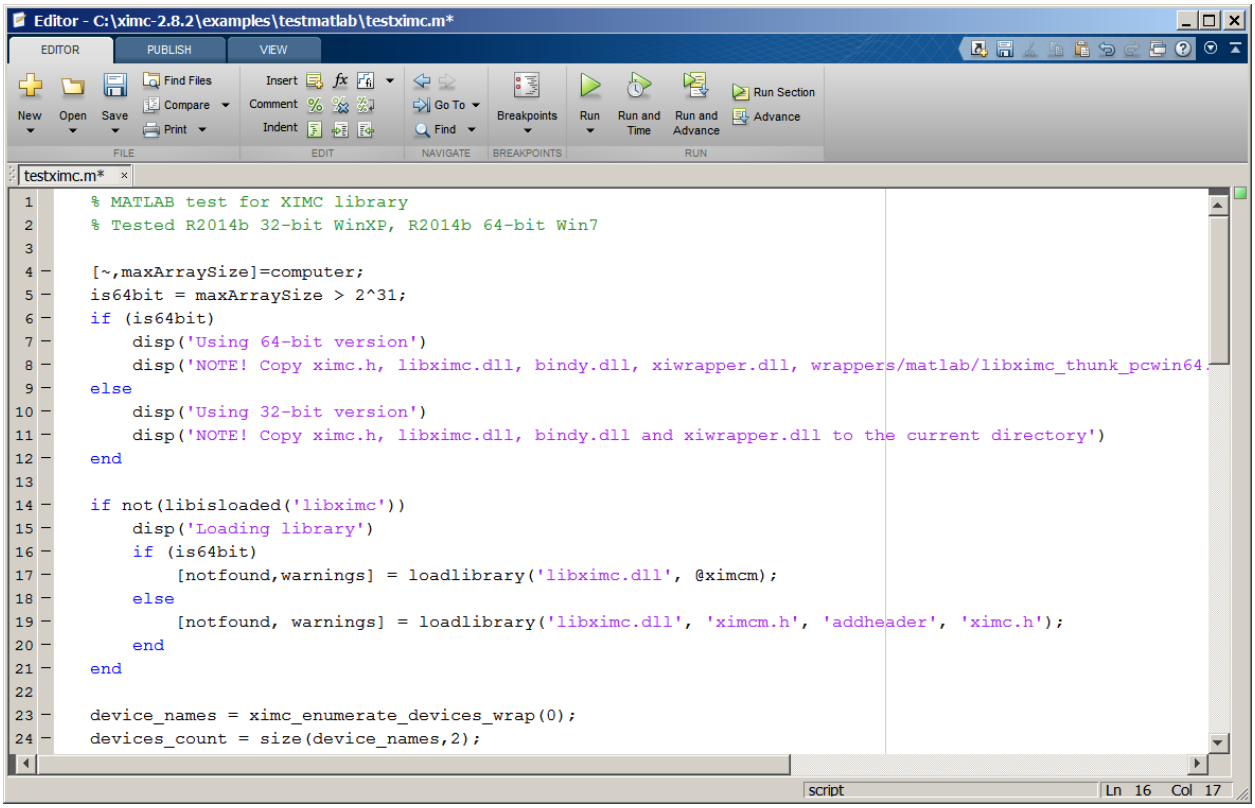
Note: Libximc library opens controllers in exclusive access mode. Any controller opened by libximc needs to be closed before it may be used by another process. Do not stop this Labview example or any other Labview program which uses libximc with the “Abort execution” button - this doesn’t give the running program a chance to call the close_device() function, thus all opened controllers will be locked and inaccessible until you close Labview environment entirely.

Warning: Since 2016, support for 8DCMC1 and 8SMC1 controllers has been discontinued by STANDA. Examples (in particular for LabVIEW) for these controllers will not be updated!

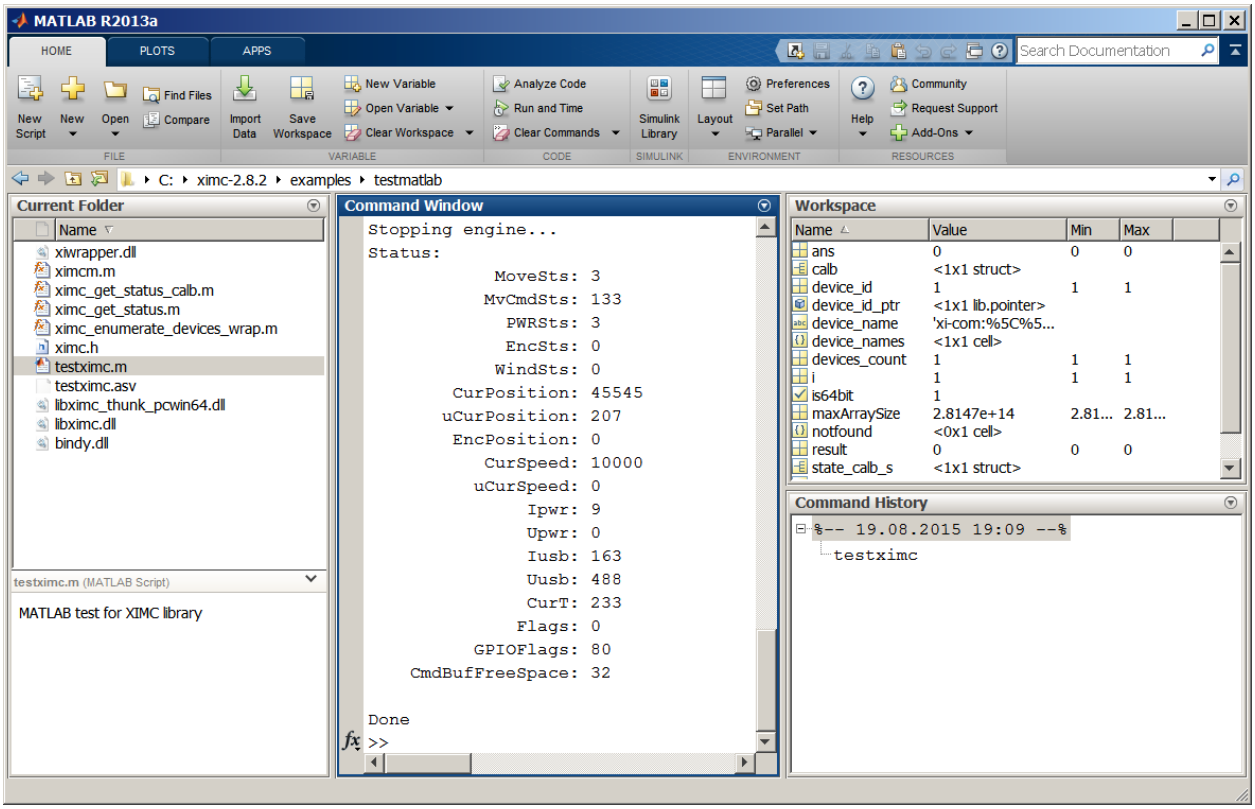
6.1.4 Working with controller in Matlab

Libximc library can be used to work with controller in Matlab.

Note: SDK requires Microsoft Visual C++ Redistributable Package 2013 (provided with SDK - vcredist_x86 or vcredist_x64).



Extract files from libximc development kit. Copy **ximc.h**, **win64\libximc.dll**, **win64\bindy.dll**, **win64\xiwrapper.dll**, **win64\wrappers\matlab\libximc_thunk_pcwin64.dll** and **win64\wrappers\matlab\ximc.m** to **examples\testmatlab** if you are using 64-bit Matlab environment or copy **ximc.h**, **win32\libximc.dll**, **win32\bindy.dll**, **win32\xiwrapper.dll** and **win32\wrappers\matlab\ximcm.h** if you are using 32-bit one. Connect the controller to the PC and run the file **testximc.m**.



In the command window you will see the output of the example which reports controller status.

You can call libximc functions from Matlab program this way: define the path to the **libximc.dll**, **bindy.dll** and **xiwrapper.dll** dynamic link libraries and its header **ximc.h** and additional to **libximc_thunk_pcwin64.dll** and **ximc.m** files if you are using 64-bin environment. Use **loadlibrary** Matlab function once to load libximc library, then use **calllib** Matlab function to call desired libximc function. You will find a list of libximc functions and their input and output parameters in the *Programming guide*.

6.1.5 Working with controller in ScanImage

Libximc library can be used to work with controller in ScanImage.

ScanImage is an open-source software application for laser scanning microscopy, electrophysiology, laser scanning photostimulation, and other physiological methods focused on neurobiology.

Standa motorized stages can now be used to create a precise setup for laser scanning. You can download the supporting software from Software page and join one of the 200 labs in the World using ScanImage.

Note: Our supporting software was tested with Windows 10 64 bit, Matlab 2017, ScanImage 2018 and DAQmx 2017.

6.1.6 Working with controller in LabWindows

Download LabWindows example from the Software page.

The archive contains two examples for LabWindows: - The console version is an example of **testcli** for the LabWindows CVI environment using the libximc library - The GUI version is an example of **testgui** with a User Interface Editor for the LabWindows CVI environment using the libximc library

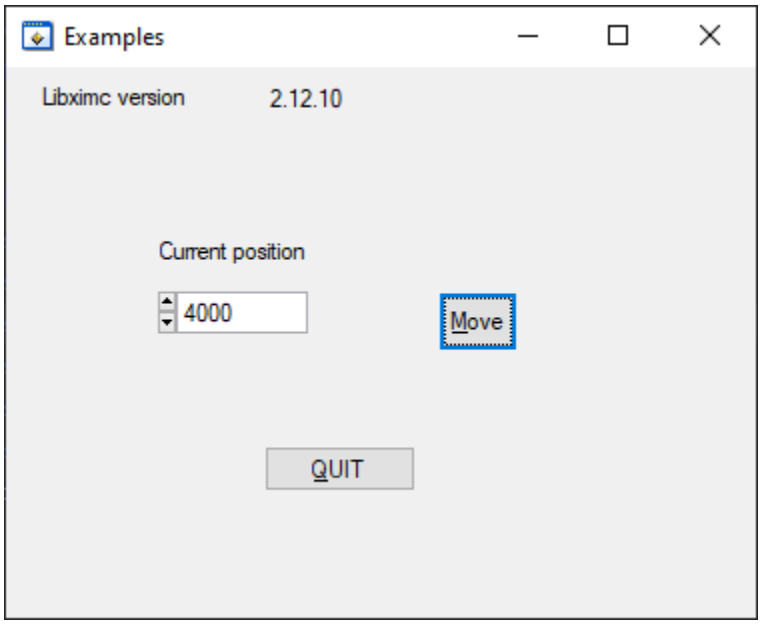
Important: The work of the examples was tested on the LabWindows CVI 2017 version.

Open the project ..\examples\testlabwindows\testcli\testcli.prj or ..\examples\testlabwindows\testgui\simple.prj in LabWindows CVI. To build, run *Build -> Rebuild*.

Copy **..ximc\win64\libximc.dll**, **..ximc\win64\bindy.dll**, **..ximc\win64\xiwrapper.dll** and **..ximc\win64\keyfile.sqlite** to the directory where the *.exe* file is located.

Note: By default, the examples have a 64-bit build installed, and the 64-bit libximc library is connected to the example. If you need to build a 32-bit version of the application, select one of the 32-bit builds in the *Build -> Configuration menu*. Remove the 64-bit version of the libximc.lib library from the project and connect the 32-bit one. The libximc library with dependencies is located in the ximc/win32 and ximc/win64 folders.

"simple.exe" program



LabWindows environment will start. You will see graphical user interface of the front panel of the example. At the top of the window you will see the libximc version. The *"Current position"* field displays position in *steps/s*. By changing the value in the field and clicking the *Move* button the move to position command will be executed.

Note: Libximc library opens controllers in exclusive access mode. Any controller opened by libximc needs to be closed before it may be used by another process. Do not stop this LabWindows example or any other LabWindows program which uses libximc with the "Close" button - this doesn't give the running program a chance to call the close_device() function, thus all opened controllers will be locked and inaccessible until you close Labwindows environment entirely.

"testcli-64.exe" program



LabWindows console will start. You will see an example of the console version. The program *"testcli"* reports the version of the library used, and also indicates which port it holds. After that, the *"testcli"* program sends the command_movr to the controller. After successful execution of the command, the close_device command will be called

---

Note: set_bindy_key("keyfile.sqlite"); Must be called before any call to *"enumerate_devices"* or *"open_device"* if you wish to use network-attached controllers. Accepts both absolute and relative paths, relative paths are resolved relative to the process working directory. If you do not need network devices then "set_bindy_key" is optional.

---

https://youtu.be/cgjVJGma-mU

Demonstration of the execution of the examples

6.1.7 A short description of the work with supported by programming languages

- *Visual C++*
- *.NET (C# and Visual Basic)*
- *Delphi*
- *Matlab*
- *Java*
- *Python*

Library usage can be examinated from test application *testapp*. Non-C languages are supported because library supports *stdcall* calling convention and so can be used with a variety of languages.

C test project is located at *"examples/testapp"* directory, C# test project - at *"xamples/testcs"*, VB.NET - *"examples/testvbnet"*, Delphi 6 - *"examples/testdelphi"*, sample bindings for MATLAB - *"examples/testmatlab"*, for Java

- *"examples/testjava"*, for Python - *"examples/testpython"*. Development kit also contains precompiled examples: *testapp* and *testappeasy* as **32** and **64-bit** applications for **Windows** and **64-bit** application for **OSX**, *testcs*, *testvbnet*, *testdelphi* - **32-bit only**, *testjava* is architecture-independent, **testmatlab and testpython are runtime-interpreted**. Also the programming guide can be downloaded from this link.

Note: SDK requires Microsoft Visual C++ Redistributable Package 2013 (provided with SDK - vcredist_x86 or vcredist_x64)

6.1.7.1 Visual C++

Testapp can be built using *testapp.sln*. Library must be compiled with MS Visual C++ too, **mingw-library isn't supported**. Make sure that Microsoft Visual C++ Redistributable Package 2013 is installed.

Open solution *examples/testapp/testapp.sln*, build and run from the IDE.

6.1.7.2 .NET (C# and Visual Basic)

Wrapper assembly for *libximc.dll* is *wrappers/csharp/ximcnet.dll*. It is provided with two different architectures and depends on .NET 2.0.

Test .NET applications for Visual Studio 2013 is located at *testcs* (*for C#*) and *testvbnet* (*for VB.NET*) respectively. Open solutions and build.

6.1.7.3 Delphi

Wrapper for *libximc.dll* is a unit *wrappers/delphi/ximc.pas* Console test application for is located at *"testdelphi"*. *Tested with Delphi 6 and only 32-bit version*. Just compile, place DLL near the executable and run program.

6.1.7.4 Matlab

Sample MATLAB program *testximc.m* is provided at the directory *examples/testmatlab*.

Before launch:

On OS X: copy *ximc/macosx/libximc.framework*, *ximc/macosx/ wrappers/ximcm.h*, *ximc/ximc.h* to the directory *examples/ matlab*. Install XCode compatible with Matlab.

On Linux: install *libximc*deb* and *libximc-dev*dev* of target architecture. Then copy *ximc/macosx/wrappers/ximcm.h* to the directory *examples/matlab*. Install gcc compatible with Matlab.

For XCode and gcc version compability check document or similar.

On Windows before the start nothing needs to be done. Change current directory in the MATLAB to the *examples/matlab*. Then launch in MATLAB prompt:

6.1.7.5 Java

How to run example on Linux. Navigate to *ximc-2.x.x./examples/testjava/compiled/* and run:

```
$ cp /usr/share/libximc/keyfile.sqlite
$ java -cp /usr/share/java/libjximc.jar:testjava.jar ru.ximc.TestJava
```

How to run example on Windows or Mac. Navigate to *ximc-2.x.x./ examples/testjava/compiled/*. Copy contents of *ximc-2.x.x/ximc/win64* or *ximc-2.x.x/ximc/macosx* accordingly to the current directory. Then run:

```
$ java -classpath libjximc.jar -classpath testjava.jar ru.ximc.TestJava
```

How to modify and recompile an example. Navigate to *examples/ testjava/compiled*. Sources are embedded in a *testjava.jar*. Extract them:

```
$ jar xvf testjava.jar ru META-INF
```

Then rebuild sources:

```
$ javac -classpath /usr/share/java/libjximc.jar -Xlint ru/ximc/TestJava.java
```

or for windows or mac:

```
$ javac -classpath libjximc.jar -Xlint ru/ximc/TestJava.java
```

Then build a jar:

```
$ jar cmf META-INF/MANIFEST.MF testjava.jar ru
```

6.1.7.6 Python

Change current directory to the *examples/testpython*. Before launch:

On OS X: copy library *ximc/macosx/libximc.framework* to the current directory.

On Linux: you may need to set *LD_LIBRARY_PATH* so Python can locate libraries with *RPATH*. For example, you may need:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:`pwd`
```

On Windows before the start nothing needs to be done. Launch Python 2 or Python 3:

```
python testpython.py
```

Note: Generic logging facility. If you want to turn on file logging, you should run the program that uses libximc library with the "XILOG" environment variable set to desired file name. This file will be opened for writing on the first log event and will be closed when the program which uses libximc terminates. Data which is sent to/received from the controller is logged along with port open and close events.

Note: Required permissions: libximc generally does not require special permissions to work, it only needs read/write access to USB-serial ports on the system. An exception to this rule is a Windows-only "fix_usbser_sys()" function - it needs elevation and will produce null result if run as a regular user.

Note: C-profiles. C-profiles are header files distributed with the libximc library. They enable one to set all controller settings for any of the supported stages with a single function call in a C/C++ program. You may see how to use C-profiles in "testcprofile" example directory.

The development kit can be downloaded on the Software page. It contains the compiled libximc library for Windows, Linux and Mac OS systems, the programming guide and the examples. Libximc is a cross-platform library that supports C++, C#, Delphi, Visual Basic, Matlab, Java and Python languages. The examples included in the library package are intended for quick acquaintance with the programming for XIMC controllers. The Libximc sources are also available for download.

The examples for working with LabView are provided independently.

Attention: Programming guide is included in libximc 2.X.X archive, where 2.X.X is the version number. It is located in /ximc-2.X.X/ximc/doc-en/libximc7-en.pdf. Also the libximc manual can be downloaded from this link. The programming guide is Doxygen-based.

6.2 Communication protocol specification

Communication protocol v20.5

- *Protocol description*
- *Command execution*
- *Controller-side error processing*
  - *Wrong command or data*
  - *CRC calculation*
  - *Transmission errors*
  - *Timeout resynchronization*
  - *Zero byte resynchronization*
- *Library-side error processing*
  - *Library return codes*
  - *Zero byte synchronization procedure*
- *Controller error response types*
  - *ERRC*
  - *ERRD*
  - *ERRV*
- *All controller commands*
  - *Command GACC*
  - *Command GBRK*
  - *Command GCAL*
  - *Command GCTL*
  - *Command GCTP*
  - *Command GEAS*
  - *Command GEDS*
  - *Command GEIO*
  - *Command GEMF*
  - *Command GENG*
  - *Command GENI*
  - *Command GENS*

- *Command GENT*
- *Command GEST*
- *Command GFBS*
- *Command GGRI*
- *Command GGRS*
- *Command GHOM*
- *Command GHSI*
- *Command GHSS*
- *Command GJOY*
- *Command GMOV*
- *Command GMTI*
- *Command GMTS*
- *Command GNME*
- *Command GNMF*
- *Command GNVM*
- *Command GPID*
- *Command GPWR*
- *Command GSEC*
- *Command GSNI*
- *Command GSNO*
- *Command GSTI*
- *Command GSTS*
- *Command GURT*
- *Command SACC*
- *Command SBRK*
- *Command SCAL*
- *Command SCTL*
- *Command SCTP*
- *Command SEAS*
- *Command SEDS*
- *Command SEIO*
- *Command SEMF*
- *Command SENG*
- *Command SENI*
- *Command SENS*

- *Command SENT*
- *Command SEST*
- *Command SFBS*
- *Command SGRI*
- *Command SGRS*
- *Command SHOM*
- *Command SHSI*
- *Command SHSS*
- *Command SJOY*
- *Command SMOV*
- *Command SMTI*
- *Command SMTS*
- *Command SNME*
- *Command SNMF*
- *Command SNVM*
- *Command SPID*
- *Command SPWR*
- *Command SSEC*
- *Command SSNI*
- *Command SSNO*
- *Command SSTI*
- *Command SSTS*
- *Command SURT*
- *Command ASIA*
- *Command CLFR*
- *Command CONN*
- *Command DBGR*
- *Command DBGW*
- *Command DISC*
- *Command EERD*
- *Command EESV*
- *Command GBLV*
- *Command GETC*
- *Command GETI*
- *Command GETM*

- *Command GETS*
- *Command GFWV*
- *Command GOFW*
- *Command GPOS*
- *Command GSER*
- *Command GUID*
- *Command HASF*
- *Command HOME*
- *Command IRND*
- *Command LEFT*
- *Command LOFT*
- *Command MOVE*
- *Command MOVR*
- *Command PWOF*
- *Command RDAN*
- *Command READ*
- *Command RERS*
- *Command REST*
- *Command RIGT*
- *Command SARS*
- *Command SAVE*
- *Command SPOS*
- *Command SSER*
- *Command SSTP*
- *Command STMS*
- *Command STOP*
- *Command UPDF*
- *Command WDAT*
- *Command WKEY*
- *Command ZERO*

6.2.1 Protocol description

Controller can be controlled from the PC using serial connection (COM-port). COM-port parameters are fixed controller-side:

- Speed: 115200 baud
- Frame size: 8 bits

- Stop-bits: 2 bits
- Parity: none
- Flow control: none
- Byte receive timeout: 400 ms
- Bit order: little endian
- Byte order: little endian

6.2.2 Command execution

All data transfers are initiated by the PC, meaning that the controller waits for incoming commands and replies accordingly. Each command is followed by the controller response, with rare exceptions of some service commands. One should not send another command without waiting for the previous command answer.

Commands are split into service, general control and general information types. Commands are executed immediately. Parameters which are set by Sxxx commands are applied no later than 1ms after acknowledgement. Command processing does not affect real-time engine control (PWM, encoder readout, etc).

Both controller and PC have an IO buffer. Received commands and command data are processed once and then removed from buffer. Each command consists of 4-byte identifier and optionally a data section followed by its 2-byte CRC. Data can be transmitted in both directions, from PC to the controller and vice versa. Command is scheduled for execution if it is a legitimate command and (in case of data) if its CRC matches. After processing a correct command controller replies with 4 bytes - the name of processed command, followed by data and its 2-byte CRC, if the command is supposed to return data.

6.2.3 Controller-side error processing

6.2.3.1 Wrong command or data

If the controller receives a command that cannot be interpreted as a legitimate command, then controller ignores this command, replies with an “errc” string and sets “command error” flag in the current status data structure. If the unreconized command contained additional data, then it can be interpreted as new command(s). In this case resynchronization is required.

If the controller receives a valid command with data and its CRC doesn’t match the CRC computed by the controller, then controller ignores this command, replies with an “errd” string and sets “data error” flag in the current status data structure. In this case synchronization is not needed.

6.2.3.2 CRC calculation

CRC is calculated for data only, 4-byte command identifier is not included. CRC algorithm in C is as follows:

```
unsigned short CRC16(INT8U *pbuf, unsigned short n)
{
  unsigned short crc, i, j, carry_flag, a;
  crc = 0xffff;
  for(i = 0; i < n; i++)
  {
    crc = crc ^ pbuf[i];
    for(j = 0; j < 8; j++)
    {
      a = crc;
      carry_flag = a & 0x0001;
      crc = crc >> 1;
      if ( carry_flag == 1 ) crc = crc ^ 0xa001;
    }
```

```
  }
  return crc;
}
```

This function receives a pointer to the data array, pbuf, and data length in bytes, n. It returns a two byte CRC code.

The example of CRC calculation:

Command code (CMD): “home” or 0x656D6F68

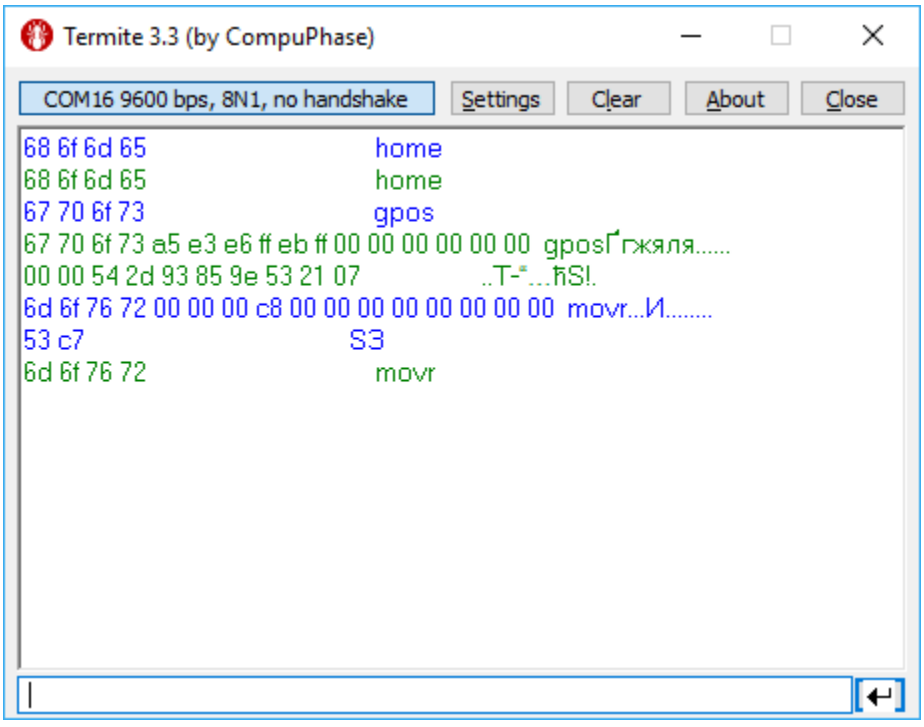
```
0x68 0x6F 0x6D 0x65
CMD
```

Command code (CMD): “gpos” or 0x736F7067

```
0x67 0x70 0x6F 0x73
CMD
```

Command code (CMD): “movr” or 0x72766F6D

```
0x6D 0x6F 0x76 0x72  0x00 0x00 0x00 0xC8  0x00 0x00  0x00 0x00 0x00 0x00 0x00 0x00
↪0x53 0xc7
CMD                  DeltaPosition        uDPos      Reserved
↪CRC
```



6.2.3.3 Transmission errors

Most probable transmission errors are missing, extra or altered byte. In usual settings transmission errors happen rarely, if at all.

Frequent errors are possible when using low-quality or broken USB-cable or board interconnection cable. Protocol is not designed for use in noisy environments and in rare cases an error may match a valid command code and get executed.

Missing byte, controller side “

A missing byte on the controller side leads to a timeout on the PC side. Command is considered to be sent unsuccessfully by the PC. Synchronization is momentarily disrupted and restored after a timeout.

Missing byte, PC side

A missing byte on the PC side leads to a timeout on PC side. Synchronization is maintained.

Extra byte, controller side

An extra byte received by the controller leads to one or several “errc” or “errd” responses. Command is considered to be sent unsuccessfully by the PC. Receive buffer may also contain one or several “errc” or “errd” responses. Synchronization is disrupted.

Extra byte, PC side

An extra byte received by the PC leads to an incorrectly interpreted command or CRC and an extra byte in the receive buffer. Synchronization is disrupted.

Altered byte, controller side

An altered byte received by the controller leads to one or several “errc” or “errd” responses. Command is considered to be sent unsuccessfully by the PC. Receive buffer may also contain one or several “errc” or “errd” responses. Synchronization can rarely be disrupted, but is generally maintained.

Altered byte, PC side

An altered byte received by the PC leads to an incorrectly interpreted command or CRC. Synchronization is maintained.

6.2.3.4 Timeout resynchronization

If during packet reception next byte wait time exceeds timeout value, then partially received command is ignored and receive buffer is cleared. Controller timeout should be less than PC timeout, taking into account time it takes to transmit the data.

6.2.3.5 Zero byte resynchronization

There are no command codes that start with a zero byte (‘\0’). This allows for a following synchronization procedure: controller always answers with a zero byte if the first command byte is zero, PC ignores first response byte if it is a zero byte. Then, if synchronization is disrupted on either side the following algorithm is used:

In case PC receives “errc”, “errd” or a wrong command answer code, then PC sends 4 to 250 zeroes to the controller (250 byte limit is caused by input buffer length and usage of I2C protocol, less than 4 zeroes do not guarantee successful resynchronization). During this time PC continuously reads incoming bytes from the controller until the first zero is received and stops sending and receiving right after that.

Received zero byte is likely not a part of a response to a previous command because on error PC receives “errc”/”errd” response. It is possible in rare cases, then synchronization procedure will start again. Therefore first zero byte received by the PC means that controller input buffer is already empty and will remain so until any command is sent. Right after receiving first zero byte from the controller PC is ready to transmit next command code. The rest of zero bytes in transit will be ignored because they will be received before controller response.

This completes the zero byte synchronization procedure.

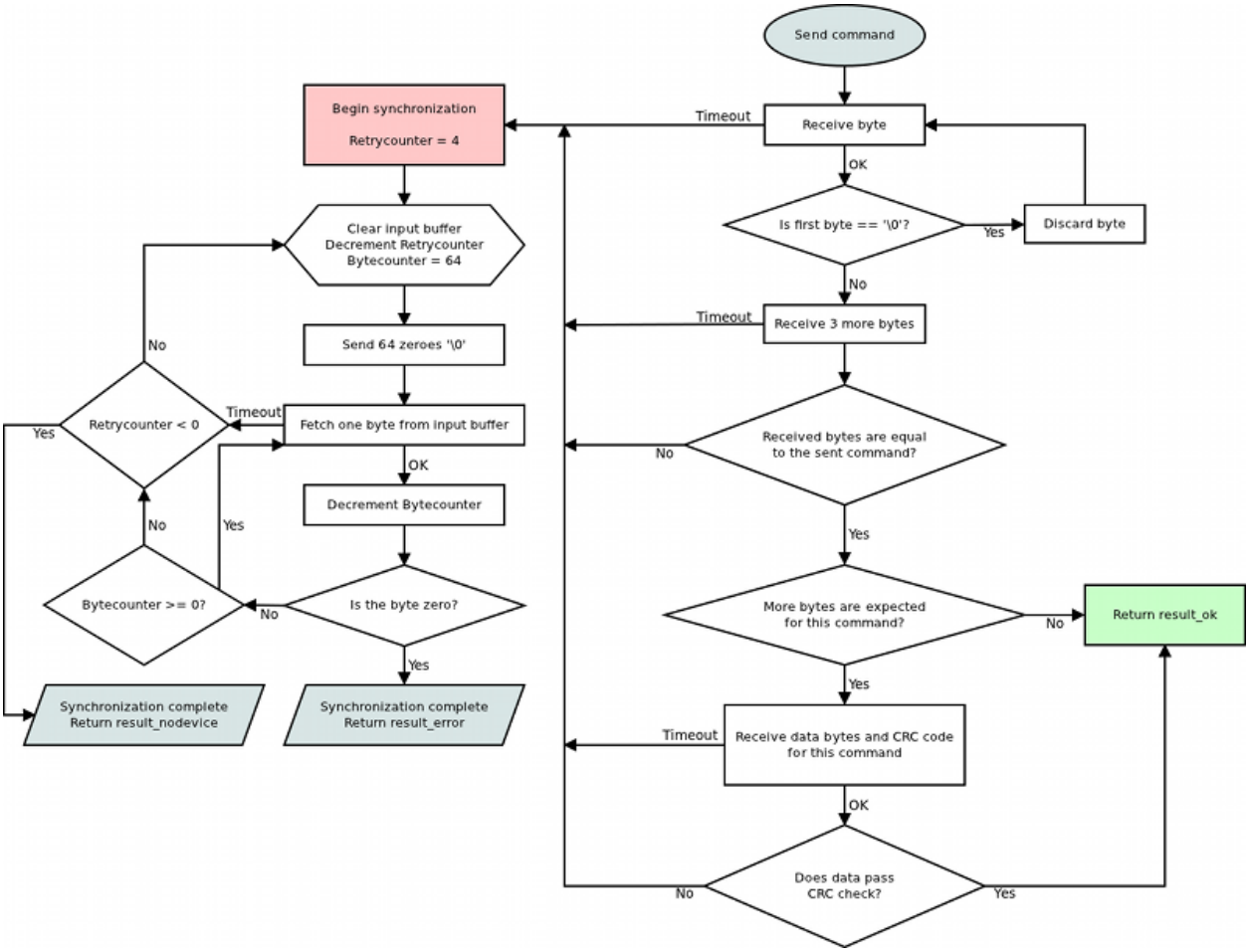
6.2.4 Library-side error processing

Nearly every library function has a return status of type *result_t*.

After sending command to the controller library reads incoming bytes until a non-zero byte is received. All zero bytes are ignored. Library reads first 4 bytes and compares them to the command code. It then waits for data section and

CRC, if needed. If first 4 received bytes do not match the sent command identifier, then zero byte synchronization procedure is launched, command is considered to be sent unsuccessfully. If first 4 received bytes match the sent command identifier and command has data section, but the received CRC doesn't match CRC calculated from the received data, then zero byte synchronization procedure is launched, command is considered to be sent unsuccessfully. If a timeout is reached while the library is waiting for the controller response, then zero byte synchronization procedure is launched, command is considered to be sent unsuccessfully.

If no errors were detected, then command is considered to be successfully completed and *result_ok* is returned.



6.2.4.1 Library return codes

- *result_ok*. No errors detected.
- *result_error*. Generic error. Can happen because of hardware problems, empty port buffer, timeout or successfull synchronization after an error. Another common reason for this error is protocol version mismatch between controller firmware and PC library.
- *result_nodevice*. Error opening device, lost connection or failed synchronization. Device reopen and/or user action is required.

If a function returns an error values of all parameters it writes to are undefined. Error code may be accompanied by detailed error description output to system log (Unix-like OS) or standard error (Windows-like OS).

6.2.4.2 Zero byte synchronization procedure

Synchronization is performed by means of sending zero ('\0') bytes and reading bytes until a zero byte is received. Optionally one may clear port buffer at the end of synchronization procedure. Initially 64 zero bytes are sent. If there were no zero bytes received during the timeout, then a string of 64 bytes is sent 3 more times. After 4 unsuccessful attempts and no zero bytes received device is considered lost. In this case library should return *result_nodevice* error code. In case of successful syncronization library returns *result_error*.

6.2.5 Controller error response types

6.2.5.1 ERRC

Answer: (4 bytes)

Code: "errc" or 0x63727265

uint32_t	errc	Command error

Description:

Controller answers with "errc" if the command is either not recognized or cannot be processed and sets the correspoding bit in status data structure.

6.2.5.2 ERRD

Answer: (4 bytes)

Code: "errd" or 0x64727265

uint32_t	errd	Data error

Description:

Controller answers with "errd" if the CRC of the data section computed by the controller doesn't match the received CRC field and sets the correspoding bit in status data structure.

6.2.5.3 ERRV

Answer: (4 bytes)

Code: "errv" or 0x76727265

uint32_t	errv	Value error

Description:

Controller answers with "errv" if any of the values in the command are out of acceptable range and can not be applied. Inacceptable value is replaced by a rounded, truncated or default value. Controller also sets the correspoding bit in status data structure.

6.2.6 All controller commands

6.2.6.1 Command GACC

Command code (CMD): "gacc" or 0x63636167.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (114 bytes)

uint32_t	CMD	Command
int8_t	MagneticBrakeInfo	The manufacturer and the part number of magnetic brake, the maximum string length is 24 characters.
float	MBRatedVoltage	Rated voltage for controlling the magnetic brake (B). Data type: float.
float	MBRatedCurrent	Rated current for controlling the magnetic brake (A). Data type: float.
float	MBTorque	Retention moment (mN m). Data type: float.
uint32_t	MBSettings	Flags of magnetic brake settings. This is a bit mask for bitwise operations.
	0x1 - MB_AVAILABLE	If flag is set the magnetic brake is available
	0x2 - MB_POWERED_HOLD	If this flag is set the magnetic brake is on when powered
int8_t	TemperatureSensorInfo	The manufacturer and the part number of the temperature sensor, the maximum string length: 24 characters.
float	TSMin	The minimum measured temperature (degrees Celsius) Data type: float.
float	TSMax	The maximum measured temperature (degrees Celsius) Data type: float.
float	TSGrad	The temperature gradient (V/degrees Celsius). Data type: float.
uint32_t	TSSettings	Flags of temperature sensor settings. This is a bit mask for bitwise operations.
	0x7 - TS_TYPE_BITS	Bits of the temperature sensor type
	0x0 - TS_TYPE_UNKNOWN	Unknow type of sensor
	0x1 - TS_TYPE_THERMOCOUPLE	Thermocouple
	0x2 - TS_TYPE_SEMICONDUCTOR	The semiconductor temperature sensor
	0x8 - TS_AVAILABLE	If flag is set the temperature sensor is available
uint32_t	LimitSwitchesSettings	Flags of limit switches settings. This is a bit mask for bitwise operations.
	0x1 - LS_ON_SW1_AVAILABLE	If flag is set the limit switch connnected to pin SW1 is available
	0x2 - LS_ON_SW2_AVAILABLE	If flag is set the limit switch connnected to pin SW2 is available
	0x4 - LS_SW1_ACTIVE_LOW	If flag is set the limit switch connnected to pin SW1 is triggered by a low level on pin
	0x8 - LS_SW2_ACTIVE_LOW	If flag is set the limit switch connnected to pin SW2 is triggered by a low level on pin
	0x10 - LS_SHORTED	If flag is set the Limit switches is shorted
uint8_t	Reserved [24]	Reserved (24 bytes)

Continued on next page

Table 6.7 – continued from previous page

uint16_t	CRC	Checksum

Description: Read additional accessories information from EEPROM.

6.2.6.2 Command GBRK

Command code (CMD): "gbrk" or 0x6B726267.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (25 bytes)

uint32_t	CMD	Command
uint16_t	t1	Time in ms between turn on motor power and turn off brake.
uint16_t	t2	Time in ms between turn off brake and moving readiness. All moving commands will execute after this interval.
uint16_t	t3	Time in ms between motor stop and turn on brake.
uint16_t	t4	Time in ms between turn on brake and turn off motor power.
uint8_t	BrakeFlags	Flags. This is a bit mask for bitwise operations.
	0x1 - BRAKE_ENABLED	Brake control is enabled, if this flag is set.
	0x2 - BRAKE_ENG_PWROFF	Brake turns off power of step motor, if this flag is set.
uint8_t	Reserved [10]	Reserved (10 bytes)
uint16_t	CRC	Checksum

Description: Read settings of brake control.

6.2.6.3 Command GCAL

Command code (CMD): "gcal" or 0x6C616367.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (118 bytes)

uint32_t	CMD	Command
float	CSS1_A	Scaling factor for the analogue measurements of the winding A current.
float	CSS1_B	Shift factor for the analogue measurements of the winding A current.
float	CSS2_A	Scaling factor for the analogue measurements of the winding B current.

Continued on next page

Table 6.11 – continued from previous page

float	CSS2_B	Shift factor for the analogue measurements of the winding B current.
float	FullCurrent_A	Scaling factor for the analogue measurements of the full current.
float	FullCurrent_B	Shift factor for the analogue measurements of the full current.
uint8_t	Reserved [88]	Reserved (88 bytes)
uint16_t	CRC	Checksum

Description: Read calibration settings. This function fill structure with calibration settings.

6.2.6.4 Command GCTL

Command code (CMD): "gctl" or 0x6C746367.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (93 bytes)

uint32_t	CMD	Command
uint32_t	MaxSpeed	Array of speeds (full step) using with joystick and button control. Range: 0..100000.
uint8_t	uMaxSpeed	Array of speeds (in microsteps) using with joystick and button control. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint16_t	Timeout	timeout[i] is time in ms, after that max_speed[i+1] is applying. It is using with buttons control only.
uint16_t	MaxClickTime	Maximum click time (in ms). Prior to the expiration of this time the first speed isn't enabled.
uint16_t	Flags	Flags. This is a bit mask for bitwise operations.
	0x3 - CONTROL_MODE_BITS	Bits to control engine by joystick or buttons.
	0x0 - CONTROL_MODE_OFF	Control is disabled.
	0x1 - CONTROL_MODE_JOY	Control by joystick.
	0x2 - CONTROL_MODE_LR	Control by left/right buttons.
	0x4 - CONTROL_BTN_LEFT_PUSHED_OPEN	Pushed left button corresponds to open contact, if this flag is set.
	0x8 - CONTROL_BTN_RIGHT_PUSHED_OPEN	Pushed right button corresponds to open contact, if this flag is set.
int32_t	DeltaPosition	Shift (delta) of position (full step)

Continued on next page

Table 6.13 – continued from previous page

int16_t	uDeltaPosition	Fractional part of the shift in micro steps. Is only used with stepper motor. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint8_t	Reserved [9]	Reserved (9 bytes)
uint16_t	CRC	Checksum

Description: Read settings of motor control. When choosing CTL_MODE = 1 switches motor control with the joystick. In this mode, the joystick to the maximum engine tends Move at MaxSpeed [i], where i = 0 if the previous use This mode is not selected another i. Buttons switch the room rate i. When CTL_MODE = 2 is switched on motor control using the Left / right. When you click on the button motor starts to move in the appropriate direction at a speed MaxSpeed [0], at the end of time Timeout [i] motor move at a speed MaxSpeed [i+1]. at Transition from MaxSpeed [i] on MaxSpeed [i +1] to acceleration, as usual.

6.2.6.5 Command GCTP

Command code (CMD): "gctp" or 0x70746367.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (18 bytes)

uint32_t	CMD	Command
uint8_t	CTPMinError	Minimum contrast steps from step motor encoder position, wich set STATE_CTP_ERROR flag. Measured in steps step motor.
uint8_t	CTPFlags	Flags. This is a bit mask for bitwise operations.
	0x1 - CTP_ENABLED	Position control is enabled, if flag set.
	0x2 - CTP_BASE	Position control is based on revolution sensor, if this flag is set; otherwise it is based on encoder.
	0x4 - CTP_ALARM_ON_ERROR	Set ALARM on mismatch, if flag set.
	0x8 - REV_SENS_INV	Sensor is active when it 0 and invert makes active level 1. That is, if you do not invert, it is normal logic - 0 is the activation.
	0x10 - CTP_ERROR_CORRECTION	Correct errors which appear when slippage if the flag is set. It works only with the encoder. Incompatible with flag CTP_ALARM_ON_ERROR.
uint8_t	Reserved [10]	Reserved (10 bytes)
uint16_t	CRC	Checksum

Description: Read settings of control position(is only used with stepper motor). When controlling the step motor with encoder (CTP_BASE 0) it is possible to detect the loss of steps. The controller knows the number of steps per revolution (GENG :: StepsPerRev) and the encoder resolution (GFBS :: IPT). When the control (flag CTP_ENABLED),

the controller stores the current position in the footsteps of SM and the current position of the encoder. Further, at each step of the position encoder is converted into steps and if the difference is greater CTPMinError, a flag STATE_CTP_ERROR. When controlling the step motor with speed sensor (CTP_BASE 1), the position is controlled by him. The active edge of input clock controller stores the current value of steps. Further, at each turn checks how many steps shifted. When a mismatch CTPMinError a flag STATE_CTP_ERROR.

6.2.6.6 Command GEAS

Command code (CMD): "geas" or 0x73616567.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (54 bytes)

uint32_t	CMD	Command
uint16_t	stepcloseloop_Kw	Mixing ratio of the actual and set speed, range [0, 100], default value 50.
uint16_t	stepcloseloop_Kp_low	Position feedback in the low-speed zone, range [0, 65535], default value 1000.
uint16_t	stepcloseloop_Kp_high	Position feedback in the high-speed zone, range [0, 65535], default value 33.
uint8_t	Reserved [42]	Reserved (42 bytes)
uint16_t	CRC	Checksum

Description: Read engine advansed settings.

6.2.6.7 Command GEDS

Command code (CMD): "geds" or 0x73646567.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (26 bytes)

uint32_t	CMD	Command
uint8_t	BorderFlags	Border flags, specify types of borders and motor behaviour on borders. This is a bit mask for bitwise operations.
	0x1 - BORDER_IS_ENCODER	Borders are fixed by predetermined encoder values, if set; borders position on limit switches, if not set.
	0x2 - BORDER_STOP_LEFT	Motor should stop on left border.
	0x4 - BORDER_STOP_RIGHT	Motor should stop on right border.
	0x8 - BORDERS_SWAP_MISSET_DETECTION	Motor should stop on both borders. Need to save motor then wrong border settings is set

Continued on next page

Table 6.19 – continued from previous page

uint8_t	EnderFlags	Ender flags, specify electrical behaviour of limit switches like order and pulled positions. This is a bit mask for bitwise operations.
	0x1 - ENDER_SWAP	First limit switch on the right side, if set; otherwise on the left side.
	0x2 - ENDER_SW1_ACTIVE_LOW	1 - Limit switch connnected to pin SW1 is triggered by a low level on pin.
	0x4 - ENDER_SW2_ACTIVE_LOW	1 - Limit switch connnected to pin SW2 is triggered by a low level on pin.
int32_t	LeftBorder	Left border position, used if BORDER_IS_ENCODER flag is set.
int16_t	uLeftBorder	Left border position in microsteps(used with stepper motor only). Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
int32_t	RightBorder	Right border position, used if BORDER_IS_ENCODER flag is set.
int16_t	uRightBorder	Right border position in microsteps. Used with stepper motor only. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint8_t	Reserved [6]	Reserved (6 bytes)
uint16_t	CRC	Checksum

Description: Read border and limit switches settings.

6.2.6.8 Command GEIO

Command code (CMD): “geio” or 0x6F696567.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (18 bytes)

uint32_t	CMD	Command
uint8_t	EXTIOSetupFlags	Configuration flags of the external I-O. This is a bit mask for bitwise operations.
	0x1 - EXTIO_SETUP_OUTPUT	EXTIO works as output if flag is set, works as input otherwise.
	0x2 - EXTIO_SETUP_INVERT	Interpret EXTIO states and fronts inverted if flag is set. Falling front as input event and low logic level as active state.
uint8_t	EXTIOModeFlags	Flags mode settings external I-O. This is a bit mask for bitwise operations.

Continued on next page

Table 6.21 – continued from previous page

	0xf - EXTIO_SETUP_MODE_IN_BITS	Bits of the behaviour selector when the signal on input goes to the active state.
	0x0 - EXTIO_SETUP_MODE_IN_NOP	Do nothing.
	0x1 - EXTIO_SETUP_MODE_IN_STOP	Issue STOP command, ceasing the engine movement.
	0x2 - EXTIO_SETUP_MODE_IN_PWOF	Issue PWOF command, powering off all engine windings.
	0x3 - EXTIO_SETUP_MODE_IN_MOVR	Issue MOVR command with last used settings.
	0x4 - EXTIO_SETUP_MODE_IN_HOME	Issue HOME command.
	0x5 - EXTIO_SETUP_MODE_IN_ALARM	Set Alarm when the signal goes to the active state.
	0xf0 - EXTIO_SETUP_MODE_OUT_BITS	Bits of the output behaviour selection.
	0x0 - EXTIO_SETUP_MODE_OUT_OFF	EXTIO pin always set in inactive state.
	0x10 - EXTIO_SETUP_MODE_OUT_ON	EXTIO pin always set in active state.
	0x20 - EXTIO_SETUP_MODE_OUT_MOVING	EXTIO pin stays active during moving state.
	0x30 - EXTIO_SETUP_MODE_OUT_ALARM	EXTIO pin stays active during Alarm state.
	0x40 - EXTIO_SETUP_MODE_OUT_MOTOR_ON	EXTIO pin stays active when windings are powered.
uint8_t	Reserved [10]	Reserved (10 bytes)
uint16_t	CRC	Checksum

Description: Read EXTIO settings. This function reads a structure with a set of EXTIO settings from controller's memory.

6.2.6.9 Command GEMF

Command code (CMD): "gemf" or 0x666D6567.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (48 bytes)

uint32_t	CMD	Command
float	L	The inductance of the windings of the motor.
float	R	The resistance of the windings of the motor.
float	Km	Electromechanical ratio of the motor.
uint8_t	BackEMFFlags	Flags of auto-settings of stepper motor. This is a bit mask for bitwise operations.
	0x1 - BACK_EMF_INDUCTANCE_AUTO	Flag of auto-detection of inductance of windings of the engine.
	0x2 - BACK_EMF_RESISTANCE_AUTO	Flag of auto-detection of resistance of windings of the engine.
	0x4 - BACK_EMF_KM_AUTO	Flag of auto-detection of electromechanical coefficient of the engine.

Continued on next page

Table 6.23 – continued from previous page

uint8_t	Reserved [29]	Reserved (29 bytes)
uint16_t	CRC	Checksum

Description: Read electromechanical settings. The settings are different for different stepper motors.

6.2.6.10 Command GENG

Command code (CMD): "geng" or 0x676E6567.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (34 bytes)

uint32_t	CMD	Command
uint16_t	NomVoltage	Rated voltage in tens of mV. Controller will keep the voltage drop on motor below this value if ENGINE_LIMIT_VOLT flag is set (used with DC only).
uint16_t	NomCurrent	Rated current (in mA). Controller will keep current consumed by motor below this value if ENGINE_LIMIT_CURR flag is set. Range: 15..8000
uint32_t	NomSpeed	Nominal (maximum) speed (in whole steps/s or rpm for DC and stepper motor as a master encoder). Controller will keep motor shaft RPM below this value if ENGINE_LIMIT_RPM flag is set. Range: 1..100000.
uint8_t	uNomSpeed	The fractional part of a nominal speed in microsteps (is only used with stepper motor). Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint16_t	EngineFlags	Set of flags specify motor shaft movement algorithm and list of limitations. This is a bit mask for bitwise operations.
	0x1 - ENGINE_REVERSE	Reverse flag. It determines motor shaft rotation direction that corresponds to feedback counts increasing. If not set (default), motor shaft rotation direction under positive voltage corresponds to feedback counts increasing and vice versa. Change it if you see that positive directions on motor and feedback are opposite.

Continued on next page

Table 6.25 – continued from previous page

	0x2 - ENGINE_CURRENT_AS_RMS	Engine current meaning flag. If the flag is unset, then engine current value is interpreted as maximum amplitude value. If the flag is set, then engine current value is interpreted as root mean square current value (for stepper) or as the current value calculated from the maximum heat dissipation (bldc).
	0x4 - ENGINE_MAX_SPEED	Max speed flag. If it is set, engine uses maximum speed achievable with the present engine settings as nominal speed.
	0x8 - ENGINE_ANTIPLAY	Play compensation flag. If it set, engine makes backlash (play) compensation procedure and reach the predetermined position accurately on low speed.
	0x10 - ENGINE_ACCEL_ON	Acceleration enable flag. If it set, motion begins with acceleration and ends with deceleration.
	0x20 - ENGINE_LIMIT_VOLT	Maximum motor voltage limit enable flag(is only used with DC motor).
	0x40 - ENGINE_LIMIT_CURR	Maximum motor current limit enable flag(is only used with DC motor).
	0x80 - ENGINE_LIMIT_RPM	Maximum motor speed limit enable flag.
int16_t	Antiplay	Number of pulses or steps for backlash (play) compensation procedure. Used if ENGINE_ANTIPLAY flag is set.
uint8_t	MicrostepMode	Settings of microstep mode(Used with stepper motor only). Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings). This is a bit mask for bitwise operations.
	0x1 - MICROSTEP_MODE_FULL	Full step mode.
	0x2 - MICROSTEP_MODE_FRAC_2	1/2 step mode.
	0x3 - MICROSTEP_MODE_FRAC_4	1/4 step mode.
	0x4 - MICROSTEP_MODE_FRAC_8	1/8 step mode.
	0x5 - MICROSTEP_MODE_FRAC_16	1/16 step mode.
	0x6 - MICROSTEP_MODE_FRAC_32	1/32 step mode.
	0x7 - MICROSTEP_MODE_FRAC_64	1/64 step mode.
	0x8 - MICROSTEP_MODE_FRAC_128	1/128 step mode.
	0x9 - MICROSTEP_MODE_FRAC_256	1/256 step mode.
uint16_t	StepsPerRev	Number of full steps per revolution(Used with stepper motor only). Range: 1..65535.
uint8_t	Reserved [12]	Reserved (12 bytes)
uint16_t	CRC	Checksum

Description: Read engine settings. This function fill structure with set of useful motor settings stored in controller's memory. These settings specify motor shaft movement algorithm, list of limitations and rated characteristics.

6.2.6.11 Command GENI

Command code (CMD): "geni" or 0x696E6567.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (70 bytes)

uint32_t	CMD	Command
int8_t	Manufacturer	Manufacturer. Max string length: 16 chars.
int8_t	PartNumber	Series and PartNumber. Max string length: 24 chars.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Description: Read encoder information from EEPROM.

6.2.6.12 Command GENS

Command code (CMD): "gens" or 0x736E6567.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (54 bytes)

uint32_t	CMD	Command
float	MaxOperatingFrequency	Max operation frequency (kHz). Data type: float.
float	SupplyVoltageMin	Minimum supply voltage (V). Data type: float.
float	SupplyVoltageMax	Maximum supply voltage (V). Data type: float.
float	MaxCurrentConsumption	Max current consumption (mA). Data type: float.
uint32_t	PPR	The number of counts per revolution
uint32_t	EncoderSettings	Encoder settings flags. This is a bit mask for bitwise operations.
	0x1 - ENCSET_DIFFERENTIAL_OUTPUT	If flag is set the encoder has differential output, else single ended output
	0x4 - ENCSET_PUSHPULL_OUTPUT	If flag is set the encoder has push-pull output, else open drain output
	0x10 - ENCSET_INDEXCHANNEL_PRESENT	If flag is set the encoder has index channel, else encoder hasn't it
	0x40 - ENCSET_REVOLUTIONSENSOR_PRESENT	If flag is set the encoder has revolution sensor, else encoder hasn't it
	0x100 - ENCSET_REVOLUTIONSENSOR_ACTIVE_HIGH	If flag is set the revolution sensor active state is high logic state, else active state is low logic state
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Description: Read encoder settings from EEPROM.

6.2.6.13 Command GENT

Command code (CMD): “gent” or 0x746E6567.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (14 bytes)

uint32_t	CMD	Command
uint8_t	EngineType	Engine type. This is a bit mask for bitwise operations.
	0x0 - ENGINE_TYPE_NONE	A value that shouldn’t be used.
	0x1 - ENGINE_TYPE_DC	DC motor.
	0x2 - ENGINE_TYPE_2DC	2 DC motors.
	0x3 - ENGINE_TYPE_STEP	Step motor.
	0x4 - ENGINE_TYPE_TEST	Duty cycle are fixed. Used only manufacturer.
	0x5 - ENGINE_TYPE_BRUSHLESS	Brushless motor.
uint8_t	DriverType	Driver type. This is a bit mask for bitwise operations.
	0x1 - DRIVER_TYPE_DISCRETE_FET	Driver with discrete FET keys. Default option.
	0x2 - DRIVER_TYPE_INTEGRATE	Driver with integrated IC.
	0x3 - DRIVER_TYPE_EXTERNAL	External driver.
uint8_t	Reserved [6]	Reserved (6 bytes)
uint16_t	CRC	Checksum

Description: Return engine type and driver type.

6.2.6.14 Command GEST

Command code (CMD): “gest” or 0x74736567.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (46 bytes)

uint32_t	CMD	Command
uint16_t	Param1	
uint8_t	Reserved [38]	Reserved (38 bytes)
uint16_t	CRC	Checksum

Description: Read extended settings.

6.2.6.15 Command GFBS

Command code (CMD): “gfbs” or 0x73626667.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (18 bytes)

uint32_t	CMD	Command
uint16_t	IPS	The number of encoder counts per shaft revolution. Range: 1..655535. The field is obsolete, it is recommended to write 0 to IPS and use the extended CountsPerTurn field. You may need to update the controller firmware to the latest version.
uint8_t	FeedbackType	Type of feedback. This is a bit mask for bitwise operations.
	0x1 - FEEDBACK_ENCODER	Feedback by encoder.
	0x4 - FEEDBACK_EMF	Feedback by EMF.
	0x5 - FEEDBACK_NONE	Feedback is absent.
	0x6 - FEEDBACK_ENCODER_MEDIATED	Feedback by encoder mediated by mechanical transmission (for example leadscrew).
uint8_t	FeedbackFlags	Flags. This is a bit mask for bitwise operations.
	0x1 - FEEDBACK_ENC_REVERSE	Reverse count of encoder.
	0xc0 - FEEDBACK_ENC_TYPE_BITS	Bits of the encoder type.
	0x0 - FEEDBACK_ENC_TYPE_AUTO	Auto detect encoder type.
	0x40 - FEEDBACK_ENC_TYPE_SINGLE_ENDED	Single ended encoder.
	0x80 - FEEDBACK_ENC_TYPE_DIFFERENTIAL	Differential encoder.
uint32_t	CountsPerTurn	The number of encoder counts per shaft revolution. Range: 1..4294967295. To use the CountsPerTurn field, write 0 in the IPS field, otherwise the value from the IPS field will be used.
uint8_t	Reserved [4]	Reserved (4 bytes)
uint16_t	CRC	Checksum

Description: Feedback settings.

6.2.6.16 Command GGRI

Command code (CMD): “ggri” or 0x69726767.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (70 bytes)

uint32_t	CMD	Command
int8_t	Manufacturer	Manufacturer. Max string length: 16 chars.
int8_t	PartNumber	Series and PartNumber. Max string length: 24 chars.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Description: Read gear information from EEPROM.

6.2.6.17 Command GGRS

Command code (CMD): "ggrs" or 0x73726767.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (58 bytes)

uint32_t	CMD	Command
float	ReductionIn	Input reduction coefficient. (Output = (ReductionOut / ReductionIn) Input) Data type: float.
float	ReductionOut	Output reduction coefficient. (Output = (ReductionOut / ReductionIn) Input) Data type: float.
float	RatedInputTorque	Max continuous torque (N m). Data type: float.
float	RatedInputSpeed	Max speed on the input shaft (rpm). Data type: float.
float	MaxOutputBacklash	Output backlash of the reduction gear(degree). Data type: float.
float	InputInertia	Equivalent input gear inertia (g cm2). Data type: float.
float	Efficiency	Reduction gear efficiency (%). Data type: float.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Description: Read gear settings from EEPROM.

6.2.6.18 Command GHOM

Command code (CMD): "ghom" or 0x6D6F6867.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (33 bytes)

uint32_t	CMD	Command
uint32_t	FastHome	Speed used for first motion (full steps). Range: 0..100000.
uint8_t	uFastHome	Part of the speed for first motion, microsteps. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint32_t	SlowHome	Speed used for second motion (full steps). Range: 0..100000.
uint8_t	uSlowHome	Part of the speed for second motion, microsteps. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
int32_t	HomeDelta	Distance from break point (full steps).
int16_t	uHomeDelta	Part of the delta distance, microsteps. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint16_t	HomeFlags	Set of flags specify direction and stopping conditions. This is a bit mask for bitwise operations.
	0x1 - HOME_DIR_FIRST	Flag defines direction of 1st motion after execution of home command. Direction is right, if set; otherwise left.
	0x2 - HOME_DIR_SECOND	Flag defines direction of 2nd motion. Direction is right, if set; otherwise left.
	0x4 - HOME_MV_SEC_EN	Use the second phase of calibration to the home position, if set; otherwise the second phase is skipped.
	0x8 - HOME_HALF_MV	If the flag is set, the stop signals are ignored in start of second movement the first half-turn.
	0x30 - HOME_STOP_FIRST_BITS	Bits of the first stop selector.
	0x10 - HOME_STOP_FIRST_REV	First motion stops by revolution sensor.
	0x20 - HOME_STOP_FIRST_SYN	First motion stops by synchronization input.
	0x30 - HOME_STOP_FIRST_LIM	First motion stops by limit switch.
	0xc0 - HOME_STOP_SECOND_BITS	Bits of the second stop selector.
	0x40 - HOME_STOP_SECOND_REV	Second motion stops by revolution sensor.
	0x80 - HOME_STOP_SECOND_SYN	Second motion stops by synchronization input.
	0xc0 - HOME_STOP_SECOND_LIM	Second motion stops by limit switch.
	0x100 - HOME_USE_FAST	Use the fast algorithm of calibration to the home position, if set; otherwise the traditional algorithm.
uint8_t	Reserved [9]	Reserved (9 bytes)
uint16_t	CRC	Checksum

Description: Read home settings. This function fill structure with settings of calibrating position.

6.2.6.19 Command GHSI

Command code (CMD): “ghsi” or 0x69736867.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (70 bytes)

uint32_t	CMD	Command
int8_t	Manufacturer	Manufacturer. Max string length: 16 chars.
int8_t	PartNumber	Series and PartNumber. Max string length: 24 chars.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Description: Read hall sensor information from EEPROM.

6.2.6.20 Command GHSS

Command code (CMD): “ghss” or 0x73736867.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (50 bytes)

uint32_t	CMD	Command
float	MaxOperatingFrequency	Max operation frequency (kHz). Data type: float.
float	SupplyVoltageMin	Minimum supply voltage (V). Data type: float.
float	SupplyVoltageMax	Maximum supply voltage (V). Data type: float.
float	MaxCurrentConsumption	Max current consumption (mA). Data type: float.
uint32_t	PPR	The number of counts per revolution
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Description: Read hall sensor settings from EEPROM.

6.2.6.21 Command GJOY

Command code (CMD): “gjoy” or 0x796F6A67.

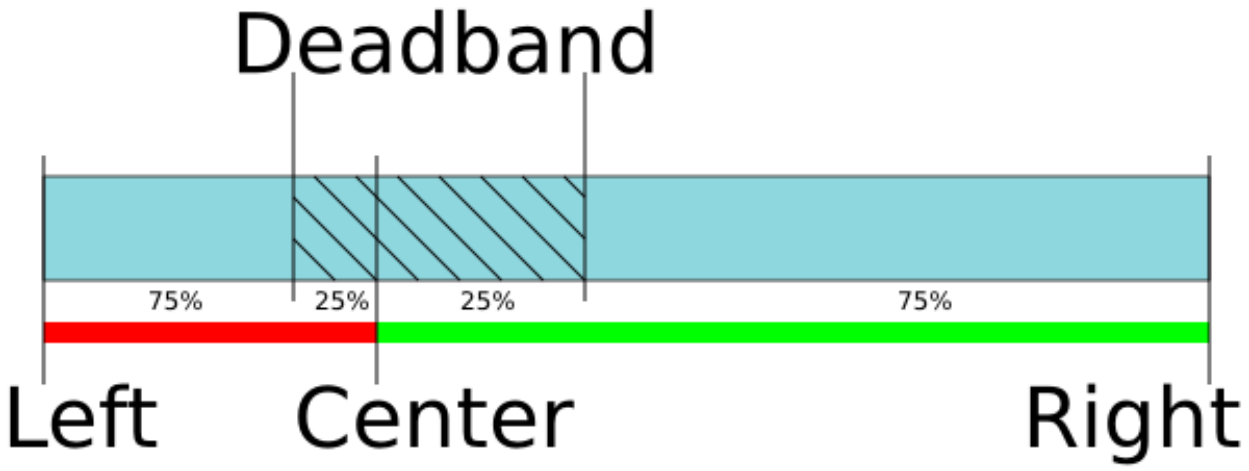
Request: (4 bytes)

uint32_t	CMD	Command

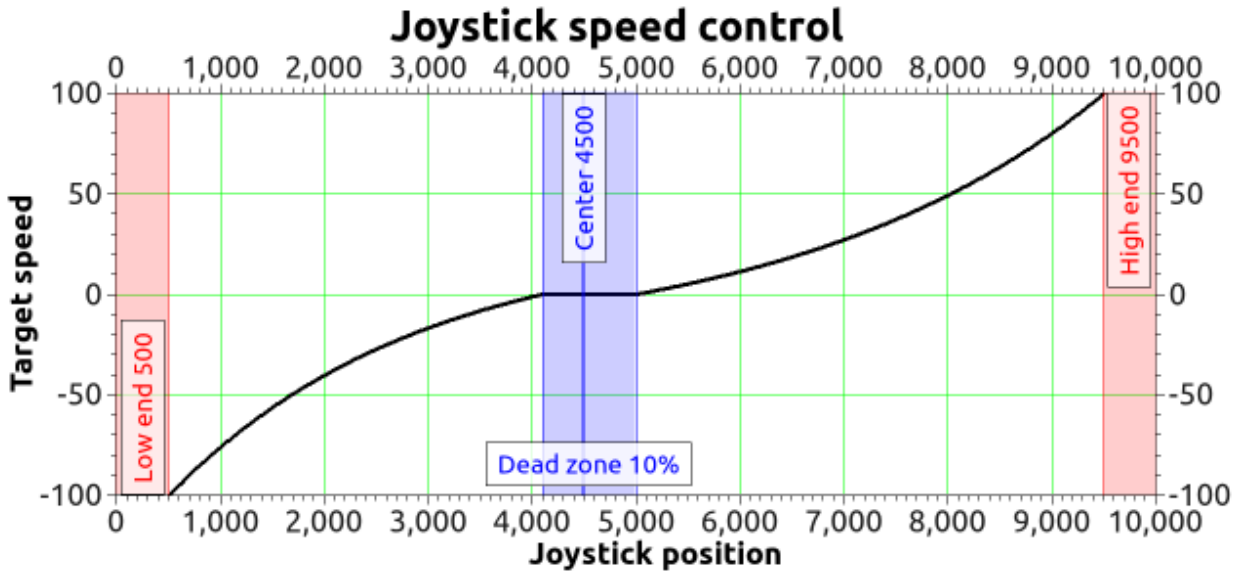
Answer: (22 bytes)

uint32_t	CMD	Command
uint16_t	JoyLowEnd	Joystick lower end position. Range: 0..10000.
uint16_t	JoyCenter	Joystick center position. Range: 0..10000.
uint16_t	JoyHighEnd	Joystick higher end position. Range: 0..10000.
uint8_t	ExpFactor	Exponential nonlinearity factor.
uint8_t	DeadZone	Joystick dead zone.
uint8_t	JoyFlags	Joystick control flags. This is a bit mask for bitwise operations.
	0x1 - JOY_REVERSE	Joystick action is reversed. Joystick deviation to the upper values correspond to negative speeds and vice versa.
uint8_t	Reserved [7]	Reserved (7 bytes)
uint16_t	CRC	Checksum

Description: Read settings of joystick. If joystick position is outside DeadZone limits from the central position a movement with speed, defined by the joystick DeadZone edge to 100% deviation, begins. Joystick positions inside DeadZone limits correspond to zero speed (soft stop of motion) and positions beyond Low and High limits correspond MaxSpeed [i] or -MaxSpeed [i] (see command SCTL), where i = 0 by default and can be changed with left/right buttons (see command SCTL). If next speed in list is zero (both integer and microstep parts), the button press is ignored. First speed in list shouldn't be zero. The DeadZone ranges are illustrated on the following picture.



The relationship between the deviation and the rate is exponential, allowing no switching speed combine high mobility and accuracy. The following picture illustrates this:



The nonlinearity parameter is adjustable. Setting it to zero makes deviation/speed relation linear.

6.2.6.22 Command GMOV

Command code (CMD): "gmov" or 0x766F6D67.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (30 bytes)

uint32_t	CMD	Command
uint32_t	Speed	Target speed (for stepper motor: steps/s, for DC: rpm). Range: 0..100000.
uint8_t	uSpeed	Target speed in microstep fractions/s. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings). Using with stepper motor only.
uint16_t	Accel	Motor shaft acceleration, steps/s^2(stepper motor) or RPM/s(DC). Range: 1..65535.
uint16_t	Decel	Motor shaft deceleration, steps/s^2(stepper motor) or RPM/s(DC). Range: 1..65535.
uint32_t	AntiplaySpeed	Speed in antiplay mode, full steps/s(stepper motor) or RPM(DC). Range: 0..100000.

Continued on next page

Table 6.49 – continued from previous page

uint8_t	uAntiplaySpeed	Speed in antiplay mode, microsteps/s. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings). Used with stepper motor only.
uint8_t	MoveFlags	Flags that control movement settings. This is a bit mask for bitwise operations.
	0x1 - RPM_DIV_1000	This flag indicates that the operating speed specified in the command is set in milli rpm. Applicable only for ENCODER feedback mode and only for BLDC motors.
uint8_t	Reserved [9]	Reserved (9 bytes)
uint16_t	CRC	Checksum

Description: Read command setup movement (speed, acceleration, threshold and etc).

6.2.6.23 Command GMTI

Command code (CMD): "gmti" or 0x69746D67.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (70 bytes)

uint32_t	CMD	Command
int8_t	Manufacturer	Manufacturer. Max string length: 16 chars.
int8_t	PartNumber	Series and PartNumber. Max string length: 24 chars.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Description: Read motor information from EEPROM.

6.2.6.24 Command GMTS

Command code (CMD): "gmts" or 0x73746D67.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (112 bytes)

uint32_t	CMD	Command
uint8_t	MotorType	Motor type. This is a bit mask for bitwise operations.

Continued on next page

Table 6.53 – continued from previous page

	0x0 - MOTOR_TYPE_UNKNOWN	Unknown type of engine
	0x1 - MOTOR_TYPE_STEP	Step engine
	0x2 - MOTOR_TYPE_DC	DC engine
	0x3 - MOTOR_TYPE_BLDC	BLDC engine
uint8_t	ReservedField	Reserved
uint16_t	Poles	Number of pole pairs for DC or BLDC motors or number of steps per rotation for stepper motor.
uint16_t	Phases	Number of phases for BLDC motors.
float	NominalVoltage	Nominal voltage on winding (B). Data type: float
float	NominalCurrent	Maximum direct current in winding for DC and BLDC engines, nominal current in windings for stepper motor (A). Data type: float.
float	NominalSpeed	Not used. Nominal speed(rpm). Used for DC and BLDC engine. Data type: float.
float	NominalTorque	Nominal torque(mN m). Used for DC and BLDC engine. Data type: float.
float	NominalPower	Nominal power(W). Used for DC and BLDC engine. Data type: float.
float	WindingResistance	Resistance of windings for DC engine, each of two windings for stepper motor or each of there windings for BLDC engine(Ohm). Data type: float.
float	WindingInductance	Inductance of windings for DC engine, each of two windings for stepper motor or each of there windings for BLDC engine(mH). Data type: float.
float	RotorInertia	Rotor inertia(g cm2). Data type: float.
float	StallTorque	Torque hold position for a stepper motor or torque at a motionless rotor for other types of engines (mN m). Data type: float.
float	DetentTorque	Holding torque position with un-powered coils (mN m). Data type: float.
float	TorqueConstant	Torque constant, which determines the aspect ratio of maximum moment of force from the rotor current flowing in the coil (mN m / A). Used mainly for DC motors. Data type: float.
float	SpeedConstant	Velocity constant, which determines the value or amplitude of the induced voltage on the motion of DC or BLDC motor (rpm / V) or stepper motor (steps/s / V). Data type: float.
float	SpeedTorqueGradient	Speed torque gradient (rpm / mN m). Data type: float.
float	MechanicalTimeConstant	Mechanical time constant (ms). Data type: float.

Continued on next page

Table 6.53 – continued from previous page

float	MaxSpeed	The maximum speed for stepper motors (steps/s) or DC and BLDC motors (rmp). Data type: float.
float	MaxCurrent	The maximum current in the winding (A). Data type: float.
float	MaxCurrentTime	Safe duration of overcurrent in the winding (ms). Data type: float.
float	NoLoadCurrent	The current consumption in idle mode (A). Used for DC and BLDC motors. Data type: float.
float	NoLoadSpeed	Idle speed (rpm). Used for DC and BLDC motors. Data type: float.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Description: Read motor settings from EEPROM.

6.2.6.25 Command GNME

Command code (CMD): "gnme" or 0x656D6E67.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (30 bytes)

uint32_t	CMD	Command
int8_t	PositionerName	User positioner name. Can be set by user for his/her convinience. Max string length: 16 chars.
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Description: Read user stage name from EEPROM.

6.2.6.26 Command GNMF

Command code (CMD): "gnmf" or 0x666D6E67.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (30 bytes)

uint32_t	CMD	Command
int8_t	ControllerName	User conroller name. Can be set by user for his/her convinience. Max string length: 16 chars.

Continued on next page

Table 6.57 – continued from previous page

uint8_t	CtrlFlags	Internal controller settings. This is a bit mask for bitwise operations.
	0x1 - EEPROM_PRECEDENCE	If the flag is set settings from external EEPROM override controller settings.
uint8_t	Reserved [7]	Reserved (7 bytes)
uint16_t	CRC	Checksum

Description: Read user controller name and flags of setting from FRAM.

6.2.6.27 Command GNVM

Command code (CMD): "gnvm" or 0x6D766E67.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (36 bytes)

uint32_t	CMD	Command
uint32_t	UserData	User data. Can be set by user for his/her convinience. Each element of the array stores only 32 bits of user data. This is important on systems where an int type contains more than 4 bytes. For example that all amd64 systems.
uint8_t	Reserved [2]	Reserved (2 bytes)
uint16_t	CRC	Checksum

Description: Read userdata from FRAM.

6.2.6.28 Command GPID

Command code (CMD): "gpid" or 0x64697067.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (48 bytes)

uint32_t	CMD	Command
uint16_t	KpU	Proportional gain for voltage PID routine
uint16_t	KiU	Integral gain for voltage PID routine
uint16_t	KdU	Differential gain for voltage PID routine
float	Kpf	Proportional gain for BLDC position PID routine
float	Kif	Integral gain for BLDC position PID routine
float	Kdf	Differential gain for BLDC position PID routine

Continued on next page

Table 6.61 – continued from previous page

uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Description: Read PID settings. This function fill structure with set of motor PID settings stored in controller's memory. These settings specify behaviour of PID routine for positioner. These factors are slightly different for different positioners. All boards are supplied with standard set of PID setting on controller's flash memory.

6.2.6.29 Command GPWR

Command code (CMD): "gpwr" or 0x72777067.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (20 bytes)

uint32_t	CMD	Command
uint8_t	HoldCurrent	Current in holding regime, percent of nominal. Range: 0..100.
uint16_t	CurrReductDelay	Time in ms from going to STOP state to reducting current.
uint16_t	PowerOffDelay	Time in s from going to STOP state to turning power off.
uint16_t	CurrentSetTime	Time in ms to reach nominal current.
uint8_t	PowerFlags	Flags with parameters of power control. This is a bit mask for bitwise operations.
	0x1 - POWER_REDUCT_ENABLED	Current reduction enabled after CurrReductDelay, if this flag is set.
	0x2 - POWER_OFF_ENABLED	Power off enabled after PowerOffDelay, if this flag is set.
	0x4 - POWER_SMOOTH_CURRENT	Current ramp-up/down is performed smoothly during current_set_time, if this flag is set.
uint8_t	Reserved [6]	Reserved (6 bytes)
uint16_t	CRC	Checksum

Description: Read settings of step motor power control. Used with stepper motor only.

6.2.6.30 Command GSEC

Command code (CMD): "gsec" or 0x63657367.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (28 bytes)

uint32_t	CMD	Command

Continued on next page

Table 6.65 – continued from previous page

uint16_t	LowUpwrOff	Lower voltage limit to turn off the motor, tens of mV.
uint16_t	CriticalIpwr	Maximum motor current which triggers ALARM state, in mA.
uint16_t	CriticalUpwr	Maximum motor voltage which triggers ALARM state, tens of mV.
uint16_t	CriticalT	Maximum temperature, which triggers ALARM state, in tenths of degrees Celcius.
uint16_t	CriticalIusb	Maximum USB current which triggers ALARM state, in mA.
uint16_t	CriticalUusb	Maximum USB voltage which triggers ALARM state, tens of mV.
uint16_t	MinimumUusb	Minimum USB voltage which triggers ALARM state, tens of mV.
uint8_t	Flags	Critical parameter flags. This is a bit mask for bitwise operations.
	0x1 - ALARM_ON_DRIVER_OVERHEATING	If this flag is set enter Alarm state on driver overheat signal.
	0x2 - LOW_UPWR_PROTECTION	If this flag is set turn off motor when voltage is lower than LowUpwrOff.
	0x4 - H_BRIDGE_ALERT	If this flag is set then turn off the power unit with a signal problem in one of the transistor bridge.
	0x8 - ALARM_ON_BORDERS_SWAP_MISSET	If this flag is set enter Alarm state on borders swap misset
	0x10 - ALARM_FLAGS_STICKING	If this flag is set only a STOP command can turn all alarms to 0
	0x20 - USB_BREAK_RECONNECT	If this flag is set USB brake reconnect module will be enable
	0x40 - ALARM_WINDING_MISMATCH	If this flag is set enter Alarm state when windings mismatch
	0x80 - ALARM_ENGINE_RESPONSE	If this flag is set enter Alarm state on response of the engine control action
uint8_t	Reserved [7]	Reserved (7 bytes)
uint16_t	CRC	Checksum

Description: Read protection settings.

6.2.6.31 Command GSNI

Command code (CMD): “gsni” or 0x696E7367.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (28 bytes)

uint32_t	CMD	Command

Continued on next page

Table 6.67 – continued from previous page

uint8_t	SyncInFlags	Input synchronization flags. This is a bit mask for bitwise operations.
	0x1 - SYNCIN_ENABLED	Synchronization in mode is enabled, if this flag is set.
	0x2 - SYNCIN_INVERT	Trigger on falling edge if flag is set, on rising edge otherwise.
	0x4 - SYNCIN_GOTOPOSITION	The engine is go to position specified in Position and uPosition, if this flag is set. And it is shift on the Position and uPosition, if this flag is unset
uint16_t	ClutterTime	Input synchronization pulse dead time (mks).
int32_t	Position	Desired position or shift (full steps)
int16_t	uPosition	The fractional part of a position or shift in microsteps. Is used with stepper motor. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint32_t	Speed	Target speed (for stepper motor: steps/s, for DC: rpm). Range: 0..100000.
uint8_t	uSpeed	Target speed in microsteps/s. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings). Using with stepper motor only.
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Description: Read input synchronization settings. This function fill structure with set of input synchronization settings, modes, periods and flags, that specify behaviour of input synchronization. All boards are supplied with standard set of these settings.

6.2.6.32 Command GSNO

Command code (CMD): "gsno" or 0x6F6E7367.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (16 bytes)

uint32_t	CMD	Command
uint8_t	SyncOutFlags	Output synchronization flags. This is a bit mask for bitwise operations.
	0x1 - SYNCOUT_ENABLED	Synchronization out pin follows the synchronization logic, if set. It governed by SYNCOUT_STATE flag otherwise.

Continued on next page

Table 6.69 – continued from previous page

	0x2 - SYNCOUT_STATE	When output state is fixed by negative SYNCOUT_ENABLED flag, the pin state is in accordance with this flag state.
	0x4 - SYNCOUT_INVERT	Low level is active, if set, and high level is active otherwise.
	0x8 - SYNCOUT_IN_STEPS	Use motor steps/encoder pulses instead of milliseconds for output pulse generation if the flag is set.
	0x10 - SYNCOUT_ONSTART	Generate synchronization pulse when movement starts.
	0x20 - SYNCOUT_ONSTOP	Generate synchronization pulse when movement stops.
	0x40 - SYNCOUT_ONPERIOD	Generate synchronization pulse every SyncOutPeriod encoder pulses.
uint16_t	SyncOutPulseSteps	This value specifies duration of output pulse. It is measured microseconds when SYNCOUT_IN_STEPS flag is cleared or in encoder pulses or motor steps when SYNCOUT_IN_STEPS is set.
uint16_t	SyncOutPeriod	This value specifies number of encoder pulses or steps between two output synchronization pulses when SYNCOUT_ONPERIOD is set.
uint32_t	Accuracy	This is the neighborhood around the target coordinates, which is getting hit in the target position and the momentum generated by the stop.
uint8_t	uAccuracy	This is the neighborhood around the target coordinates in microsteps (only used with stepper motor). Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint16_t	CRC	Checksum

Description: Read output synchronization settings. This function fill structure with set of output synchronization settings, modes, periods and flags, that specify behaviour of output synchronization. All boards are supplied with standard set of these settings.

6.2.6.33 Command GSTI

Command code (CMD): "gsti" or 0x69747367.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (70 bytes)

uint32_t	CMD	Command

Continued on next page

Table 6.71 – continued from previous page

int8_t	Manufacturer	Manufacturer. Max string length: 16 chars.
int8_t	PartNumber	Series and PartNumber. Max string length: 24 chars.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Description: Read stage information from EEPROM.

6.2.6.34 Command GSTS

Command code (CMD): "gsts" or 0x73747367.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (70 bytes)

uint32_t	CMD	Command
float	LeadScrewPitch	Lead screw pitch (mm). Data type: float.
int8_t	Units	Units for MaxSpeed and TravelRange fields of the structure (steps, degrees, mm, . . .). Max string length: 8 chars.
float	MaxSpeed	Max speed (Units/c). Data type: float.
float	TravelRange	Travel range (Units). Data type: float.
float	SupplyVoltageMin	Supply voltage minimum (V). Data type: float.
float	SupplyVoltageMax	Supply voltage maximum (V). Data type: float.
float	MaxCurrentConsumption	Max current consumption (A). Data type: float.
float	HorizontalLoadCapacity	Horizontal load capacity (kg). Data type: float.
float	VerticalLoadCapacity	Vertical load capacity (kg). Data type: float.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Description: Read stage settings from EEPROM.

6.2.6.35 Command GURT

Command code (CMD): "gurt" or 0x74727567.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (16 bytes)

uint32_t	CMD	Command
uint32_t	Speed	UART speed (in bauds)
uint16_t	UARTSetupFlags	UART setup flags. This is a bit mask for bitwise operations.
	0x3 - UART_PARITY_BITS	Bits of the parity.
	0x0 - UART_PARITY_BIT_EVEN	Parity bit 1, if even
	0x1 - UART_PARITY_BIT_ODD	Parity bit 1, if odd
	0x2 - UART_PARITY_BIT_SPACE	Parity bit always 0
	0x3 - UART_PARITY_BIT_MARK	Parity bit always 1
	0x4 - UART_PARITY_BIT_USE	None parity
	0x8 - UART_STOP_BIT	If set - one stop bit, else two stop bit
uint8_t	Reserved [4]	Reserved (4 bytes)
uint16_t	CRC	Checksum

Description: Read UART settings. This function fill structure with UART settings.

6.2.6.36 Command SACC

Command code (CMD): “sacc” or 0x63636173.

Request: (114 bytes)

uint32_t	CMD	Command
int8_t	MagneticBrakeInfo	The manufacturer and the part number of magnetic brake, the maximum string length is 24 characters.
float	MBRatedVoltage	Rated voltage for controlling the magnetic brake (B). Data type: float.
float	MBRatedCurrent	Rated current for controlling the magnetic brake (A). Data type: float.
float	MBTorque	Retention moment (mN m). Data type: float.
uint32_t	MBSettings	Flags of magnetic brake settings. This is a bit mask for bitwise operations.
	0x1 - MB_AVAILABLE	If flag is set the magnetic brake is available
	0x2 - MB_POWERED_HOLD	If this flag is set the magnetic brake is on when powered
int8_t	TemperatureSensorInfo	The manufacturer and the part number of the temperature sensor, the maximum string length: 24 characters.
float	TSMin	The minimum measured temperature (degrees Celsius) Data type: float.
float	TSMax	The maximum measured temperature (degrees Celsius) Data type: float.
float	TSGrad	The temperature gradient (V/degrees Celsius). Data type: float.
uint32_t	TSSettings	Flags of temperature sensor settings. This is a bit mask for bitwise operations.
	0x7 - TS_TYPE_BITS	Bits of the temperature sensor type
	0x0 - TS_TYPE_UNKNOWN	Unknow type of sensor

Continued on next page

Table 6.76 – continued from previous page

	0x1 - TS_TYPE_THERMOCOUPLE	Thermocouple
	0x2 - TS_TYPE_SEMICONDUCTOR	The semiconductor temperature sensor
	0x8 - TS_AVAILABLE	If flag is set the temperature sensor is available
uint32_t	LimitSwitchesSettings	Flags of limit switches settings. This is a bit mask for bitwise operations.
	0x1 - LS_ON_SW1_AVAILABLE	If flag is set the limit switch connnected to pin SW1 is available
	0x2 - LS_ON_SW2_AVAILABLE	If flag is set the limit switch connnected to pin SW2 is available
	0x4 - LS_SW1_ACTIVE_LOW	If flag is set the limit switch connnected to pin SW1 is triggered by a low level on pin
	0x8 - LS_SW2_ACTIVE_LOW	If flag is set the limit switch connnected to pin SW2 is triggered by a low level on pin
	0x10 - LS_SHORTED	If flag is set the Limit switches is shorted
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set additional accessories information to EEPROM. Can be used by manufacturer only.

6.2.6.37 Command SBRK

Command code (CMD): “sbrk” or 0x6B726273.

Request: (25 bytes)

uint32_t	CMD	Command
uint16_t	t1	Time in ms between turn on motor power and turn off brake.
uint16_t	t2	Time in ms between turn off brake and moving readiness. All moving commands will execute after this interval.
uint16_t	t3	Time in ms between motor stop and turn on brake.
uint16_t	t4	Time in ms between turn on brake and turn off motor power.
uint8_t	BrakeFlags	Flags. This is a bit mask for bitwise operations.
	0x1 - BRAKE_ENABLED	Brake control is enabled, if this flag is set.
	0x2 - BRAKE_ENG_PWROFF	Brake turns off power of step motor, if this flag is set.
uint8_t	Reserved [10]	Reserved (10 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set settings of brake control.

6.2.6.38 Command SCAL

Command code (CMD): "scal" or 0x6C616373.

Request: (118 bytes)

uint32_t	CMD	Command
float	CSS1_A	Scaling factor for the analogue measurements of the winding A current.
float	CSS1_B	Shift factor for the analogue measurements of the winding A current.
float	CSS2_A	Scaling factor for the analogue measurements of the winding B current.
float	CSS2_B	Shift factor for the analogue measurements of the winding B current.
float	FullCurrent_A	Scaling factor for the analogue measurements of the full current.
float	FullCurrent_B	Shift factor for the analogue measurements of the full current.
uint8_t	Reserved [88]	Reserved (88 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set calibration settings. This function send structure with calibration settings to controller's memory.

6.2.6.39 Command SCTL

Command code (CMD): "sctl" or 0x6C746373.

Request: (93 bytes)

uint32_t	CMD	Command
uint32_t	MaxSpeed	Array of speeds (full step) using with joystick and button control. Range: 0..100000.
uint8_t	uMaxSpeed	Array of speeds (in microsteps) using with joystick and button control. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint16_t	Timeout	timeout[i] is time in ms, after that max_speed[i+1] is applying. It is using with buttons control only.

Continued on next page

Table 6.82 – continued from previous page

uint16_t	MaxClickTime	Maximum click time (in ms). Prior to the expiration of this time the first speed isn't enabled.
uint16_t	Flags	Flags. This is a bit mask for bitwise operations.
	0x3 - CONTROL_MODE_BITS	Bits to control engine by joystick or buttons.
	0x0 - CONTROL_MODE_OFF	Control is disabled.
	0x1 - CONTROL_MODE_JOY	Control by joystick.
	0x2 - CONTROL_MODE_LR	Control by left/right buttons.
	0x4 - CONTROL_BTN_LEFT_PUSHED_OPEN	Pushed left button corresponds to open contact, if this flag is set.
	0x8 - CONTROL_BTN_RIGHT_PUSHED_OPEN	Pushed right button corresponds to open contact, if this flag is set.
int32_t	DeltaPosition	Shift (delta) of position (full step)
int16_t	uDeltaPosition	Fractional part of the shift in micro steps. Is only used with stepper motor. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint8_t	Reserved [9]	Reserved (9 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set settings of motor control. When choosing CTL_MODE = 1 switches motor control with the joystick. In this mode, the joystick to the maximum engine tends Move at MaxSpeed [i], where i = 0 if the previous use This mode is not selected another i. Buttons switch the room rate i. When CTL_MODE = 2 is switched on motor control using the Left / right. When you click on the button motor starts to move in the appropriate direction at a speed MaxSpeed [0], at the end of time Timeout [i] motor move at a speed MaxSpeed [i+1]. at Transition from MaxSpeed [i] on MaxSpeed [i +1] to acceleration, as usual.

6.2.6.40 Command SCTP

Command code (CMD): "sctp" or 0x70746373.

Request: (18 bytes)

uint32_t	CMD	Command
uint8_t	CTPMinError	Minimum contrast steps from step motor encoder position, wich set STATE_CTP_ERROR flag. Measured in steps step motor.
uint8_t	CTPFlags	Flags. This is a bit mask for bitwise operations.
	0x1 - CTP_ENABLED	Position control is enabled, if flag set.

Continued on next page

Table 6.84 – continued from previous page

	0x2 - CTP_BASE	Position control is based on revolution sensor, if this flag is set; otherwise it is based on encoder.
	0x4 - CTP_ALARM_ON_ERROR	Set ALARM on mismatch, if flag set.
	0x8 - REV_SENS_INV	Sensor is active when it 0 and invert makes active level 1. That is, if you do not invert, it is normal logic - 0 is the activation.
	0x10 - CTP_ERROR_CORRECTION	Correct errors which appear when slippage if the flag is set. It works only with the encoder. Incompatible with flag CTP_ALARM_ON_ERROR.
uint8_t	Reserved [10]	Reserved (10 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set settings of control position(is only used with stepper motor). When controlling the step motor with encoder (CTP_BASE 0) it is possible to detect the loss of steps. The controller knows the number of steps per revolution (GENG :: StepsPerRev) and the encoder resolution (GFBS :: IPT). When the control (flag CTP_ENABLED), the controller stores the current position in the footsteps of SM and the current position of the encoder. Further, at each step of the position encoder is converted into steps and if the difference is greater CTPMinError, a flag STATE_CTP_ERROR. When controlling the step motor with speed sensor (CTP_BASE 1), the position is controlled by him. The active edge of input clock controller stores the current value of steps. Further, at each turn checks how many steps shifted. When a mismatch CTPMinError a flag STATE_CTP_ERROR.

6.2.6.41 Command SEAS

Command code (CMD): “seas” or 0x73616573.

Request: (54 bytes)

uint32_t	CMD	Command
uint16_t	stepcloseloop_Kw	Mixing ratio of the actual and set speed, range [0, 100], default value 50.
uint16_t	stepcloseloop_Kp_low	Position feedback in the low-speed zone, range [0, 65535], default value 1000.
uint16_t	stepcloseloop_Kp_high	Position feedback in the high-speed zone, range [0, 65535], default value 33.
uint8_t	Reserved [42]	Reserved (42 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set engine advansed settings.

6.2.6.42 Command SEDS

Command code (CMD): "seds" or 0x73646573.

Request: (26 bytes)

uint32_t	CMD	Command
uint8_t	BorderFlags	Border flags, specify types of borders and motor behaviour on borders. This is a bit mask for bitwise operations.
	0x1 - BORDER_IS_ENCODER	Borders are fixed by predetermined encoder values, if set; borders position on limit switches, if not set.
	0x2 - BORDER_STOP_LEFT	Motor should stop on left border.
	0x4 - BORDER_STOP_RIGHT	Motor should stop on right border.
	0x8 - BORDERS_SWAP_MISSET_DETECTION	Motor should stop on both borders. Need to save motor then wrong border settings is set
uint8_t	EnderFlags	Ender flags, specify electrical behaviour of limit switches like order and pulled positions. This is a bit mask for bitwise operations.
	0x1 - ENDER_SWAP	First limit switch on the right side, if set; otherwise on the left side.
	0x2 - ENDER_SW1_ACTIVE_LOW	1 - Limit switch connnected to pin SW1 is triggered by a low level on pin.
	0x4 - ENDER_SW2_ACTIVE_LOW	1 - Limit switch connnected to pin SW2 is triggered by a low level on pin.
int32_t	LeftBorder	Left border position, used if BORDER_IS_ENCODER flag is set.
int16_t	uLeftBorder	Left border position in microsteps(used with stepper motor only). Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
int32_t	RightBorder	Right border position, used if BORDER_IS_ENCODER flag is set.
int16_t	uRightBorder	Right border position in microsteps. Used with stepper motor only. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint8_t	Reserved [6]	Reserved (6 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set border and limit switches settings.

6.2.6.43 Command SEIO

Command code (CMD): “seio” or 0x6F696573.

Request: (18 bytes)

uint32_t	CMD	Command
uint8_t	EXTIOSetupFlags	Configuration flags of the external I-O. This is a bit mask for bitwise operations.
	0x1 - EXTIO_SETUP_OUTPUT	EXTIO works as output if flag is set, works as input otherwise.
	0x2 - EXTIO_SETUP_INVERT	Interpret EXTIO states and fronts inverted if flag is set. Falling front as input event and low logic level as active state.
uint8_t	EXTIOModeFlags	Flags mode settings external I-O. This is a bit mask for bitwise operations.
	0xf - EXTIO_SETUP_MODE_IN_BITS	Bits of the behaviour selector when the signal on input goes to the active state.
	0x0 - EXTIO_SETUP_MODE_IN_NOP	Do nothing.
	0x1 - EXTIO_SETUP_MODE_IN_STOP	Issue STOP command, ceasing the engine movement.
	0x2 - EXTIO_SETUP_MODE_IN_PWOF	Issue PWOF command, powering off all engine windings.
	0x3 - EXTIO_SETUP_MODE_IN_MOVR	Issue MOVR command with last used settings.
	0x4 - EXTIO_SETUP_MODE_IN_HOME	Issue HOME command.
	0x5 - EXTIO_SETUP_MODE_IN_ALARM	Set Alarm when the signal goes to the active state.
	0xf0 - EXTIO_SETUP_MODE_OUT_BITS	Bits of the output behaviour selection.
	0x0 - EXTIO_SETUP_MODE_OUT_OFF	EXTIO pin always set in inactive state.
	0x10 - EXTIO_SETUP_MODE_OUT_ON	EXTIO pin always set in active state.
	0x20 - EXTIO_SETUP_MODE_OUT_MOVING	EXTIO pin stays active during moving state.
	0x30 - EXTIO_SETUP_MODE_OUT_ALARM	EXTIO pin stays active during Alarm state.
	0x40 - EXTIO_SETUP_MODE_OUT_MOTOR_ON	EXTIO pin stays active when windings are powered.
uint8_t	Reserved [10]	Reserved (10 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set EXTIO settings. This function writes a structure with a set of EXTIO settings to controller’s memory. By default input event are signalled through rising front and output states are signalled by high logic state.

6.2.6.44 Command SEMF

Command code (CMD): “semf” or 0x666D6573.

Request: (48 bytes)

uint32_t	CMD	Command
float	L	The inductance of the windings of the motor.
float	R	The resistance of the windings of the motor.
float	Km	Electromechanical ratio of the motor.
uint8_t	BackEMFFlags	Flags of auto-settings of stepper motor. This is a bit mask for bitwise operations.
	0x1 - BACK_EMF_INDUCTANCE_AUTO	Flag of auto-detection of inductance of windings of the engine.
	0x2 - BACK_EMF_RESISTANCE_AUTO	Flag of auto-detection of resistance of windings of the engine.
	0x4 - BACK_EMF_KM_AUTO	Flag of auto-detection of electromechanical coefficient of the engine.
uint8_t	Reserved [29]	Reserved (29 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set electromechanical coefficients. The settings are different for different stepper motors. Please download the new settings when you change the motor.

6.2.6.45 Command SENG

Command code (CMD): "seng" or 0x676E6573.

Request: (34 bytes)

uint32_t	CMD	Command
uint16_t	NomVoltage	Rated voltage in tens of mV. Controller will keep the voltage drop on motor below this value if ENGINE_LIMIT_VOLT flag is set (used with DC only).
uint16_t	NomCurrent	Rated current (in mA). Controller will keep current consumed by motor below this value if ENGINE_LIMIT_CURR flag is set. Range: 15..8000
uint32_t	NomSpeed	Nominal (maximum) speed (in whole steps/s or rpm for DC and stepper motor as a master encoder). Controller will keep motor shaft RPM below this value if ENGINE_LIMIT_RPM flag is set. Range: 1..100000.

Continued on next page

Table 6.94 – continued from previous page

uint8_t	uNomSpeed	The fractional part of a nominal speed in microsteps (is only used with stepper motor). Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint16_t	EngineFlags	Set of flags specify motor shaft movement algorithm and list of limitations. This is a bit mask for bitwise operations.
	0x1 - ENGINE_REVERSE	Reverse flag. It determines motor shaft rotation direction that corresponds to feedback counts increasing. If not set (default), motor shaft rotation direction under positive voltage corresponds to feedback counts increasing and vice versa. Change it if you see that positive directions on motor and feedback are opposite.
	0x2 - ENGINE_CURRENT_AS_RMS	Engine current meaning flag. If the flag is unset, then engine current value is interpreted as maximum amplitude value. If the flag is set, then engine current value is interpreted as root mean square current value (for stepper) or as the current value calculated from the maximum heat dissipation (bldc).
	0x4 - ENGINE_MAX_SPEED	Max speed flag. If it is set, engine uses maximum speed achievable with the present engine settings as nominal speed.
	0x8 - ENGINE_ANTIPLAY	Play compensation flag. If it set, engine makes backlash (play) compensation procedure and reach the predetermined position accurately on low speed.
	0x10 - ENGINE_ACCEL_ON	Acceleration enable flag. If it set, motion begins with acceleration and ends with deceleration.
	0x20 - ENGINE_LIMIT_VOLT	Maximum motor voltage limit enable flag(is only used with DC motor).
	0x40 - ENGINE_LIMIT_CURR	Maximum motor current limit enable flag(is only used with DC motor).
	0x80 - ENGINE_LIMIT_RPM	Maximum motor speed limit enable flag.
int16_t	Antiplay	Number of pulses or steps for backlash (play) compensation procedure. Used if ENGINE_ANTIPLAY flag is set.
uint8_t	MicrostepMode	Settings of microstep mode(Used with stepper motor only). Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings). This is a bit mask for bitwise operations.
	0x1 - MICROSTEP_MODE_FULL	Full step mode.

Continued on next page

Table 6.94 – continued from previous page

	0x2 - MICROSTEP_MODE_FRAC_2	1/2 step mode.
	0x3 - MICROSTEP_MODE_FRAC_4	1/4 step mode.
	0x4 - MICROSTEP_MODE_FRAC_8	1/8 step mode.
	0x5 - MICROSTEP_MODE_FRAC_16	1/16 step mode.
	0x6 - MICROSTEP_MODE_FRAC_32	1/32 step mode.
	0x7 - MICROSTEP_MODE_FRAC_64	1/64 step mode.
	0x8 - MICROSTEP_MODE_FRAC_128	1/128 step mode.
	0x9 - MICROSTEP_MODE_FRAC_256	1/256 step mode.
uint16_t	StepsPerRev	Number of full steps per revolution(Used with stepper motor only). Range: 1..65535.
uint8_t	Reserved [12]	Reserved (12 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set engine settings. This function send structure with set of engine settings to controller's memory. These settings specify motor shaft movement algorithm, list of limitations and rated characteristics. Use it when you change motor, encoder, positioner etc. Please note that wrong engine settings lead to device malfunction, can lead to irreversible damage of board.

6.2.6.46 Command SENI

Command code (CMD): "seni" or 0x696E6573.

Request: (70 bytes)

uint32_t	CMD	Command
int8_t	Manufacturer	Manufacturer. Max string length: 16 chars.
int8_t	PartNumber	Series and PartNumber. Max string length: 24 chars.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set encoder information to EEPROM. Can be used by manufacturer only.

6.2.6.47 Command SENS

Command code (CMD): "sens" or 0x736E6573.

Request: (54 bytes)

uint32_t	CMD	Command

Continued on next page

Table 6.98 – continued from previous page

float	MaxOperatingFrequency	Max operation frequency (kHz). Data type: float.
float	SupplyVoltageMin	Minimum supply voltage (V). Data type: float.
float	SupplyVoltageMax	Maximum supply voltage (V). Data type: float.
float	MaxCurrentConsumption	Max current consumption (mA). Data type: float.
uint32_t	PPR	The number of counts per revolution
uint32_t	EncoderSettings	Encoder settings flags. This is a bit mask for bitwise operations.
	0x1 - ENCSET_DIFFERENTIAL_OUTPUT	If flag is set the encoder has differential output, else single ended output
	0x4 - ENCSET_PUSHPULL_OUTPUT	If flag is set the encoder has push-pull output, else open drain output
	0x10 - ENCSET_INDEXCHANNEL_PRESENT	If flag is set the encoder has index channel, else encoder hasn't it
	0x40 - ENCSET_REVOLUTIONSENSOR_PRESENT	If flag is set the encoder has revolution sensor, else encoder hasn't it
	0x100 - ENCSET_REVOLUTIONSENSOR_ACTIVE_HIGH	If flag is set the revolution sensor active state is high logic state, else active state is low logic state
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set encoder settings to EEPROM. Can be used by manufacturer only.

6.2.6.48 Command SENT

Command code (CMD): "sent" or 0x746E6573.

Request: (14 bytes)

uint32_t	CMD	Command
uint8_t	EngineType	Engine type. This is a bit mask for bitwise operations.
	0x0 - ENGINE_TYPE_NONE	A value that shouldn't be used.
	0x1 - ENGINE_TYPE_DC	DC motor.
	0x2 - ENGINE_TYPE_2DC	2 DC motors.
	0x3 - ENGINE_TYPE_STEP	Step motor.
	0x4 - ENGINE_TYPE_TEST	Duty cycle are fixed. Used only manufacturer.
	0x5 - ENGINE_TYPE_BRUSHLESS	Brushless motor.
uint8_t	DriverType	Driver type. This is a bit mask for bitwise operations.
	0x1 - DRIVER_TYPE_DISCRETE_FET	Driver with discrete FET keys. Default option.

Continued on next page

Table 6.100 – continued from previous page

	0x2 - DRIVER_TYPE_INTEGRATE	Driver with integrated IC.
	0x3 - DRIVER_TYPE_EXTERNAL	External driver.
uint8_t	Reserved [6]	Reserved (6 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set engine type and driver type.

6.2.6.49 Command SEST

Command code (CMD): "sest" or 0x74736573.

Request: (46 bytes)

uint32_t	CMD	Command
uint16_t	Param1	
uint8_t	Reserved [38]	Reserved (38 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set extended settings.

6.2.6.50 Command SFBS

Command code (CMD): "sfbs" or 0x73626673.

Request: (18 bytes)

uint32_t	CMD	Command
uint16_t	IPS	The number of encoder counts per shaft revolution. Range: 1..655535. The field is obsolete, it is recommended to write 0 to IPS and use the extended CountsPerTurn field. You may need to update the controller firmware to the latest version.
uint8_t	FeedbackType	Type of feedback. This is a bit mask for bitwise operations.
	0x1 - FEEDBACK_ENCODER	Feedback by encoder.
	0x4 - FEEDBACK_EMF	Feedback by EMF.
	0x5 - FEEDBACK_NONE	Feedback is absent.
	0x6 - FEEDBACK_ENCODER_MEDIATED	Feedback by encoder mediated by mechanical transmission (for example leadscrew).
uint8_t	FeedbackFlags	Flags. This is a bit mask for bitwise operations.

Continued on next page

Table 6.104 – continued from previous page

	0x1 - FEEDBACK_ENC_REVERSE	Reverse count of encoder.
	0xc0 - FEEDBACK_ENC_TYPE_BITS	Bits of the encoder type.
	0x0 - FEEDBACK_ENC_TYPE_AUTO	Auto detect encoder type.
	0x40 - FEEDBACK_ENC_TYPE_SINGLE_ENDED	Single ended encoder.
	0x80 - FEEDBACK_ENC_TYPE_DIFFERENTIAL	Differential encoder.
uint32_t	CountsPerTurn	The number of encoder counts per shaft revolution. Range: 1..4294967295. To use the CountsPerTurn field, write 0 in the IPS field, otherwise the value from the IPS field will be used.
uint8_t	Reserved [4]	Reserved (4 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Feedback settings.

6.2.6.51 Command SGRI

Command code (CMD): "sgri" or 0x69726773.

Request: (70 bytes)

uint32_t	CMD	Command
int8_t	Manufacturer	Manufacturer. Max string length: 16 chars.
int8_t	PartNumber	Series and PartNumber. Max string length: 24 chars.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set gear information to EEPROM. Can be used by manufacturer only.

6.2.6.52 Command SGRS

Command code (CMD): "sgrs" or 0x73726773.

Request: (58 bytes)

uint32_t	CMD	Command
float	ReductionIn	Input reduction coefficient. (Output = (ReductionOut / ReductionIn) Input) Data type: float.

Continued on next page

Table 6.108 – continued from previous page

float	ReductionOut	Output reduction coefficient. (Output = (ReductionOut / ReductionIn) Input) Data type: float.
float	RatedInputTorque	Max continuous torque (N m). Data type: float.
float	RatedInputSpeed	Max speed on the input shaft (rpm). Data type: float.
float	MaxOutputBacklash	Output backlash of the reduction gear(degree). Data type: float.
float	InputInertia	Equivalent input gear inertia (g cm2). Data type: float.
float	Efficiency	Reduction gear efficiency (%). Data type: float.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set gear settings to EEPROM. Can be used by manufacturer only.

6.2.6.53 Command SHOM

Command code (CMD): "shom" or 0x6D6F6873.

Request: (33 bytes)

uint32_t	CMD	Command
uint32_t	FastHome	Speed used for first motion (full steps). Range: 0..100000.
uint8_t	uFastHome	Part of the speed for first motion, microsteps. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint32_t	SlowHome	Speed used for second motion (full steps). Range: 0..100000.
uint8_t	uSlowHome	Part of the speed for second motion, microsteps. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
int32_t	HomeDelta	Distance from break point (full steps).
int16_t	uHomeDelta	Part of the delta distance, microsteps. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).

Continued on next page

Table 6.110 – continued from previous page

uint16_t	HomeFlags	Set of flags specify direction and stopping conditions. This is a bit mask for bitwise operations.
	0x1 - HOME_DIR_FIRST	Flag defines direction of 1st motion after execution of home command. Direction is right, if set; otherwise left.
	0x2 - HOME_DIR_SECOND	Flag defines direction of 2nd motion. Direction is right, if set; otherwise left.
	0x4 - HOME_MV_SEC_EN	Use the second phase of calibration to the home position, if set; otherwise the second phase is skipped.
	0x8 - HOME_HALF_MV	If the flag is set, the stop signals are ignored in start of second movement the first half-turn.
	0x30 - HOME_STOP_FIRST_BITS	Bits of the first stop selector.
	0x10 - HOME_STOP_FIRST_REV	First motion stops by revolution sensor.
	0x20 - HOME_STOP_FIRST_SYN	First motion stops by synchronization input.
	0x30 - HOME_STOP_FIRST_LIM	First motion stops by limit switch.
	0xc0 - HOME_STOP_SECOND_BITS	Bits of the second stop selector.
	0x40 - HOME_STOP_SECOND_REV	Second motion stops by revolution sensor.
	0x80 - HOME_STOP_SECOND_SYN	Second motion stops by synchronization input.
	0xc0 - HOME_STOP_SECOND_LIM	Second motion stops by limit switch.
	0x100 - HOME_USE_FAST	Use the fast algorithm of calibration to the home position, if set; otherwise the traditional algorithm.
uint8_t	Reserved [9]	Reserved (9 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set home settings. This function send structure with calibrating position settings to controller's memory.

6.2.6.54 Command SHSI

Command code (CMD): "shsi" or 0x69736873.

Request: (70 bytes)

uint32_t	CMD	Command
int8_t	Manufacturer	Manufacturer. Max string length: 16 chars.
int8_t	PartNumber	Series and PartNumber. Max string length: 24 chars.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set hall sensor information to EEPROM. Can be used by manufacturer only.

6.2.6.55 Command SHSS

Command code (CMD): "shss" or 0x73736873.

Request: (50 bytes)

uint32_t	CMD	Command
float	MaxOperatingFrequency	Max operation frequency (kHz). Data type: float.
float	SupplyVoltageMin	Minimum supply voltage (V). Data type: float.
float	SupplyVoltageMax	Maximum supply voltage (V). Data type: float.
float	MaxCurrentConsumption	Max current consumption (mA). Data type: float.
uint32_t	PPR	The number of counts per revolution
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set hall sensor settings to EEPROM. Can be used by manufacturer only.

6.2.6.56 Command SJOY

Command code (CMD): "sjoy" or 0x796F6A73.

Request: (22 bytes)

uint32_t	CMD	Command
uint16_t	JoyLowEnd	Joystick lower end position. Range: 0..10000.
uint16_t	JoyCenter	Joystick center position. Range: 0..10000.
uint16_t	JoyHighEnd	Joystick higher end position. Range: 0..10000.
uint8_t	ExpFactor	Exponential nonlinearity factor.
uint8_t	DeadZone	Joystick dead zone.
uint8_t	JoyFlags	Joystick control flags. This is a bit mask for bitwise operations.
	0x1 - JOY_REVERSE	Joystick action is reversed. Joystick deviation to the upper values correspond to negative speeds and vice versa.
uint8_t	Reserved [7]	Reserved (7 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set settings of joystick. If joystick position is outside DeadZone limits from the central position a movement with speed, defined by the joystick DeadZone edge to 100% deviation, begins. Joystick positions inside DeadZone limits correspond to zero speed (soft stop of motion) and positions beyond Low and High limits correspond MaxSpeed [i] or -MaxSpeed [i] (see command SCTL), where i = 0 by default and can be changed with left/right buttons (see command SCTL). If next speed in list is zero (both integer and microstep parts), the button press is ignored. First speed in list shouldn't be zero. The DeadZone ranges are illustrated on the following picture. !/attachments/download/5563/range25p.png! The relationship between the deviation and the rate is exponential, allowing no switching speed combine high mobility and accuracy. The following picture illustrates this: !/attachments/download/3092/ExpJoystick.png! The nonlinearity parameter is adjustable. Setting it to zero makes deviation/speed relation linear.

6.2.6.57 Command SMOV

Command code (CMD): "smov" or 0x766F6D73.

Request: (30 bytes)

uint32_t	CMD	Command
uint32_t	Speed	Target speed (for stepper motor: steps/s, for DC: rpm). Range: 0..100000.
uint8_t	uSpeed	Target speed in microstep fractions/s. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings). Using with stepper motor only.
uint16_t	Accel	Motor shaft acceleration, steps/s^2(stepper motor) or RPM/s(DC). Range: 1..65535.
uint16_t	Decel	Motor shaft deceleration, steps/s^2(stepper motor) or RPM/s(DC). Range: 1..65535.
uint32_t	AntiplaySpeed	Speed in antiplay mode, full steps/s(stepper motor) or RPM(DC). Range: 0..100000.
uint8_t	uAntiplaySpeed	Speed in antiplay mode, microsteps/s. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings). Used with stepper motor only.
uint8_t	MoveFlags	Flags that control movement settings. This is a bit mask for bitwise operations.

Continued on next page

Table 6.118 – continued from previous page

	0x1 - RPM_DIV_1000	This flag indicates that the operating speed specified in the command is set in milli rpm. Applicable only for ENCODER feedback mode and only for BLDC motors.
uint8_t	Reserved [9]	Reserved (9 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set command setup movement (speed, acceleration, threshold and etc).

6.2.6.58 Command SMTI

Command code (CMD): "smti" or 0x69746D73.

Request: (70 bytes)

uint32_t	CMD	Command
int8_t	Manufacturer	Manufacturer. Max string length: 16 chars.
int8_t	PartNumber	Series and PartNumber. Max string length: 24 chars.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set motor information to EEPROM. Can be used by manufacturer only.

6.2.6.59 Command SMTS

Command code (CMD): "smts" or 0x73746D73.

Request: (112 bytes)

uint32_t	CMD	Command
uint8_t	MotorType	Motor type. This is a bit mask for bitwise operations.
	0x0 - MOTOR_TYPE_UNKNOWN	Unknown type of engine
	0x1 - MOTOR_TYPE_STEP	Step engine
	0x2 - MOTOR_TYPE_DC	DC engine
	0x3 - MOTOR_TYPE_BLDC	BLDC engine
uint8_t	ReservedField	Reserved
uint16_t	Poles	Number of pole pairs for DC or BLDC motors or number of steps per rotation for stepper motor.

Continued on next page

Table 6.122 – continued from previous page

uint16_t	Phases	Number of phases for BLDC motors.
float	NominalVoltage	Nominal voltage on winding (B). Data type: float
float	NominalCurrent	Maximum direct current in winding for DC and BLDC engines, nominal current in windings for stepper motor (A). Data type: float.
float	NominalSpeed	Not used. Nominal speed(rpm). Used for DC and BLDC engine. Data type: float.
float	NominalTorque	Nominal torque(mN m). Used for DC and BLDC engine. Data type: float.
float	NominalPower	Nominal power(W). Used for DC and BLDC engine. Data type: float.
float	WindingResistance	Resistance of windings for DC engine, each of two windings for stepper motor or each of there windings for BLDC engine(Ohm). Data type: float.
float	WindingInductance	Inductance of windings for DC engine, each of two windings for stepper motor or each of there windings for BLDC engine(mH). Data type: float.
float	RotorInertia	Rotor inertia(g cm2). Data type: float.
float	StallTorque	Torque hold position for a stepper motor or torque at a motionless rotor for other types of engines (mN m). Data type: float.
float	DetentTorque	Holding torque position with un-powered coils (mN m). Data type: float.
float	TorqueConstant	Torque constant, which determines the aspect ratio of maximum moment of force from the rotor current flowing in the coil (mN m / A). Used mainly for DC motors. Data type: float.
float	SpeedConstant	Velocity constant, which determines the value or amplitude of the induced voltage on the motion of DC or BLDC motor (rpm / V) or stepper motor (steps/s / V). Data type: float.
float	SpeedTorqueGradient	Speed torque gradient (rpm / mN m). Data type: float.
float	MechanicalTimeConstant	Mechanical time constant (ms). Data type: float.
float	MaxSpeed	The maximum speed for stepper motors (steps/s) or DC and BLDC motors (rmp). Data type: float.
float	MaxCurrent	The maximum current in the winding (A). Data type: float.
float	MaxCurrentTime	Safe duration of overcurrent in the winding (ms). Data type: float.

Continued on next page

Table 6.122 – continued from previous page

float	NoLoadCurrent	The current consumption in idle mode (A). Used for DC and BLDC motors. Data type: float.
float	NoLoadSpeed	Idle speed (rpm). Used for DC and BLDC motors. Data type: float.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set motor settings to EEPROM. Can be used by manufacturer only.

6.2.6.60 Command SNME

Command code (CMD): "snme" or 0x656D6E73.

Request: (30 bytes)

uint32_t	CMD	Command
int8_t	PositionerName	User positioner name. Can be set by user for his/her convinience. Max string length: 16 chars.
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Write user stage name from EEPROM.

6.2.6.61 Command SNMF

Command code (CMD): "snmf" or 0x666D6E73.

Request: (30 bytes)

uint32_t	CMD	Command
int8_t	ControllerName	User conroller name. Can be set by user for his/her convinience. Max string length: 16 chars.
uint8_t	CtrlFlags	Internal controller settings. This is a bit mask for bitwise operations.
	0x1 - EEPROM_PRECEDENCE	If the flag is set settings from external EEPROM override controller settings.
uint8_t	Reserved [7]	Reserved (7 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Write user controller name and flags of setting from FRAM.

6.2.6.62 Command SNVM

Command code (CMD): "snvm" or 0x6D766E73.

Request: (36 bytes)

uint32_t	CMD	Command
uint32_t	UserData	User data. Can be set by user for his/her convinience. Each element of the array stores only 32 bits of user data. This is important on systems where an int type contains more than 4 bytes. For example that all amd64 systems.
uint8_t	Reserved [2]	Reserved (2 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Write userdata into FRAM.

6.2.6.63 Command SPID

Command code (CMD): "spid" or 0x64697073.

Request: (48 bytes)

uint32_t	CMD	Command
uint16_t	KpU	Proportional gain for voltage PID routine
uint16_t	KiU	Integral gain for voltage PID routine
uint16_t	KdU	Differential gain for voltage PID routine
float	Kpf	Proportional gain for BLDC position PID routine
float	Kif	Integral gain for BLDC position PID routine
float	Kdf	Differential gain for BLDC position PID routine
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set PID settings. This function send structure with set of PID factors to controller's memory. These settings specify behaviour of PID routine for positioner. These factors are slightly different for different positioners. All boards are supplied with standard set of PID setting on controller's flash memory. Please use it for loading new PID settings when you change positioner. Please note that wrong PID settings lead to device malfunction.

6.2.6.64 Command SPWR

Command code (CMD): "spwr" or 0x72777073.

Request: (20 bytes)

uint32_t	CMD	Command
uint8_t	HoldCurrent	Current in holding regime, percent of nominal. Range: 0..100.
uint16_t	CurrReductDelay	Time in ms from going to STOP state to reducting current.
uint16_t	PowerOffDelay	Time in s from going to STOP state to turning power off.
uint16_t	CurrentSetTime	Time in ms to reach nominal current.
uint8_t	PowerFlags	Flags with parameters of power control. This is a bit mask for bitwise operations.
	0x1 - POWER_REDUCT_ENABLED	Current reduction enabled after CurrReductDelay, if this flag is set.
	0x2 - POWER_OFF_ENABLED	Power off enabled after PowerOffDelay, if this flag is set.
	0x4 - POWER_SMOOTH_CURRENT	Current ramp-up/down is performed smoothly during current_set_time, if this flag is set.
uint8_t	Reserved [6]	Reserved (6 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set settings of step motor power control. Used with stepper motor only.

6.2.6.65 Command SSEC

Command code (CMD): "ssec" or 0x63657373.

Request: (28 bytes)

uint32_t	CMD	Command
uint16_t	LowUpwrOff	Lower voltage limit to turn off the motor, tens of mV.
uint16_t	CriticalIpwr	Maximum motor current which triggers ALARM state, in mA.
uint16_t	CriticalUpwr	Maximum motor voltage which triggers ALARM state, tens of mV.

Continued on next page

Table 6.134 – continued from previous page

uint16_t	CriticalT	Maximum temperature, which triggers ALARM state, in tenths of degrees Celcius.
uint16_t	CriticalIusb	Maximum USB current which triggers ALARM state, in mA.
uint16_t	CriticalUusb	Maximum USB voltage which triggers ALARM state, tens of mV.
uint16_t	MinimumUusb	Minimum USB voltage which triggers ALARM state, tens of mV.
uint8_t	Flags	Critical parameter flags. This is a bit mask for bitwise operations.
	0x1 - ALARM_ON_DRIVER_OVERHEATING	If this flag is set enter Alarm state on driver overheat signal.
	0x2 - LOW_UPWR_PROTECTION	If this flag is set turn off motor when voltage is lower than LowUpwrOff.
	0x4 - H_BRIDGE_ALERT	If this flag is set then turn off the power unit with a signal problem in one of the transistor bridge.
	0x8 - ALARM_ON_BORDERS_SWAP_MISSET	If this flag is set enter Alarm state on borders swap misset
	0x10 - ALARM_FLAGS_STICKING	If this flag is set only a STOP command can turn all alarms to 0
	0x20 - USB_BREAK_RECONNECT	If this flag is set USB brake reconnect module will be enable
	0x40 - ALARM_WINDING_MISMATCH	If this flag is set enter Alarm state when windings mismatch
	0x80 - ALARM_ENGINE_RESPONSE	If this flag is set enter Alarm state on response of the engine control action
uint8_t	Reserved [7]	Reserved (7 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set protection settings.

6.2.6.66 Command SSNI

Command code (CMD): “ssni” or 0x696E7373.

Request: (28 bytes)

uint32_t	CMD	Command
uint8_t	SyncInFlags	Input synchronization flags. This is a bit mask for bitwise operations.
	0x1 - SYNCIN_ENABLED	Synchronization in mode is enabled, if this flag is set.
	0x2 - SYNCIN_INVERT	Trigger on falling edge if flag is set, on rising edge otherwise.

Continued on next page

Table 6.136 – continued from previous page

	0x4 - SYNCIN_GOTOPOSITION	The engine is go to position specified in Position and uPosition, if this flag is set. And it is shift on the Position and uPosition, if this flag is unset
uint16_t	ClutterTime	Input synchronization pulse dead time (mks).
int32_t	Position	Desired position or shift (full steps)
int16_t	uPosition	The fractional part of a position or shift in microsteps. Is used with stepper motor. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint32_t	Speed	Target speed (for stepper motor: steps/s, for DC: rpm). Range: 0..100000.
uint8_t	uSpeed	Target speed in microsteps/s. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings). Using with stepper motor only.
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set input synchronization settings. This function send structure with set of input synchronization settings, that specify behaviour of input synchronization, to controller's memory. All boards are supplied with standard set of these settings.

6.2.6.67 Command SSNO

Command code (CMD): "ssno" or 0x6F6E7373.

Request: (16 bytes)

uint32_t	CMD	Command
uint8_t	SyncOutFlags	Output synchronization flags. This is a bit mask for bitwise operations.
	0x1 - SYNCOUT_ENABLED	Synchronization out pin follows the synchronization logic, if set. It governed by SYNCOUT_STATE flag otherwise.
	0x2 - SYNCOUT_STATE	When output state is fixed by negative SYNCOUT_ENABLED flag, the pin state is in accordance with this flag state.
	0x4 - SYNCOUT_INVERT	Low level is active, if set, and high level is active otherwise.

Continued on next page

Table 6.138 – continued from previous page

	0x8 - SYNCOUT_IN_STEPS	Use motor steps/encoder pulses instead of milliseconds for output pulse generation if the flag is set.
	0x10 - SYNCOUT_ONSTART	Generate synchronization pulse when movement starts.
	0x20 - SYNCOUT_ONSTOP	Generate synchronization pulse when movement stops.
	0x40 - SYNCOUT_ONPERIOD	Generate synchronization pulse every SyncOutPeriod encoder pulses.
uint16_t	SyncOutPulseSteps	This value specifies duration of output pulse. It is measured microseconds when SYNCOUT_IN_STEPS flag is cleared or in encoder pulses or motor steps when SYNCOUT_IN_STEPS is set.
uint16_t	SyncOutPeriod	This value specifies number of encoder pulses or steps between two output synchronization pulses when SYNCOUT_ONPERIOD is set.
uint32_t	Accuracy	This is the neighborhood around the target coordinates, which is getting hit in the target position and the momentum generated by the stop.
uint8_t	uAccuracy	This is the neighborhood around the target coordinates in microsteps (only used with stepper motor). Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set output synchronization settings. This function send structure with set of output synchronization settings, that specify behaviour of output synchronization, to controller's memory. All boards are supplied with standard set of these settings.

6.2.6.68 Command SSTI

Command code (CMD): "ssti" or 0x69747373.

Request: (70 bytes)

uint32_t	CMD	Command
int8_t	Manufacturer	Manufacturer. Max string length: 16 chars.
int8_t	PartNumber	Series and PartNumber. Max string length: 24 chars.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set stage information to EEPROM. Can be used by manufacturer only.

6.2.6.69 Command SSTS

Command code (CMD): "ssts" or 0x73747373.

Request: (70 bytes)

uint32_t	CMD	Command
float	LeadScrewPitch	Lead screw pitch (mm). Data type: float.
int8_t	Units	Units for MaxSpeed and TravelRange fields of the structure (steps, degrees, mm, . . .). Max string length: 8 chars.
float	MaxSpeed	Max speed (Units/c). Data type: float.
float	TravelRange	Travel range (Units). Data type: float.
float	SupplyVoltageMin	Supply voltage minimum (V). Data type: float.
float	SupplyVoltageMax	Supply voltage maximum (V). Data type: float.
float	MaxCurrentConsumption	Max current consumption (A). Data type: float.
float	HorizontalLoadCapacity	Horizontal load capacity (kg). Data type: float.
float	VerticalLoadCapacity	Vertical load capacity (kg). Data type: float.
uint8_t	Reserved [24]	Reserved (24 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set stage settings to EEPROM. Can be used by manufacturer only

6.2.6.70 Command SURT

Command code (CMD): "surt" or 0x74727573.

Request: (16 bytes)

uint32_t	CMD	Command
uint32_t	Speed	UART speed (in bauds)
uint16_t	UARTSetupFlags	UART setup flags. This is a bit mask for bitwise operations.
	0x3 - UART_PARITY_BITS	Bits of the parity.
	0x0 - UART_PARITY_BIT_EVEN	Parity bit 1, if even

Continued on next page

Table 6.144 – continued from previous page

	0x1 - UART_PARITY_BIT_ODD	Parity bit 1, if odd
	0x2 - UART_PARITY_BIT_SPACE	Parity bit always 0
	0x3 - UART_PARITY_BIT_MARK	Parity bit always 1
	0x4 - UART_PARITY_BIT_USE	None parity
	0x8 - UART_STOP_BIT	If set - one stop bit, else two stop bit
uint8_t	Reserved [4]	Reserved (4 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Set UART settings. This function send structure with UART settings to controller's memory.

6.2.6.71 Command ASIA

Command code (CMD): "asia" or 0x61697361.

Request: (22 bytes)

uint32_t	CMD	Command
int32_t	Position	Desired position or shift (full steps)
int16_t	uPosition	The fractional part of a position or shift in microsteps. Is only used with stepper motor. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint32_t	Time	Time for which you want to achieve the desired position in microseconds.
uint8_t	Reserved [6]	Reserved (6 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: This command adds one element of the FIFO commands that are executed when input clock pulse. Each pulse synchronization or perform that action, which is described in SSNI, if the buffer is empty, or the oldest loaded into the buffer action to temporarily replace the speed and coordinate in SSNI. In the latter case this action is erased from the buffer. The number of remaining empty buffer elements can be found in the structure of GETS.

6.2.6.72 Command CLFR

Command code (CMD): "clfr" or 0x72666C63.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: The command for clear controller FRAM. The memory is cleared by filling the whole memory byte 0x00. After cleaning, the controller restarts. No response to this command.

6.2.6.73 Command CONN

Command code (CMD): "conn" or 0x6E6E6F63.

Request: (14 bytes)

uint32_t	CMD	Command
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Answer: (15 bytes)

uint32_t	CMD	Command
uint8_t	sresult	Result of command.
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Description: Command to open a session ISP (in-system programming) when downloading the firmware. Result = RESULT_OK, if the command loader. Result = RESULT_SOFT_ERROR, if at the time of the command error occurred. Result not available using the library command_update_firmware, the field value is processed by the function.

6.2.6.74 Command DBGR

Command code (CMD): "dbgr" or 0x72676264.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (142 bytes)

uint32_t	CMD	Command
uint8_t	DebugData	Arbitrary debug data.
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Description: Read data from firmware for debug purpose. Its use depends on context, firmware version and previous history.

6.2.6.75 Command DBGW

Command code (CMD): “dbgw” or 0x77676264.

Request: (142 bytes)

uint32_t	CMD	Command
uint8_t	DebugData	Arbitrary debug data.
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Write data to firmware for debug purpose.

6.2.6.76 Command DISC

Command code (CMD): “disc” or 0x63736964.

Request: (14 bytes)

uint32_t	CMD	Command
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Answer: (15 bytes)

uint32_t	CMD	Command
uint8_t	sresult	Result of command.
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Description: Command to close the session ISP (in-system programming) when loading firmware. Result = RESULT_OK, if the command loader. Result = RESULT_HARD_ERROR, if at the time of the command hardware error occurred. Result = RESULT_SOFT_ERROR, if at the time of the command software error occurred. Result not available using the library command_update_firmware, the field value is processed by the function.

6.2.6.77 Command EERD

Command code (CMD): “eerd” or 0x64726565.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Read settings from controller’s RAM to stage’s EEPROM memory, which spontaneity connected to stage and it isn’t change without it mechanical reconstruction.

6.2.6.78 Command EESV

Command code (CMD): “eesv” or 0x76736565.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Save settings from controller’s RAM to stage’s EEPROM memory, which spontaneity connected to stage and it isn’t change without it mechanical reconstruction. Can be used by manufacturer only.

6.2.6.79 Command GBLV

Command code (CMD): “gblv” or 0x766C6267.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (10 bytes)

uint32_t	CMD	Command
uint8_t	Major	Bootloader major version number
uint8_t	Minor	Bootloader minor version number
uint16_t	Release	Bootloader release version number
uint16_t	CRC	Checksum

Description: Read controller’s firmware version.

6.2.6.80 Command GETC

Command code (CMD): “getc” or 0x63746567.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (38 bytes)

uint32_t	CMD	Command
int16_t	WindingVoltageA	In the case step motor, the voltage across the winding A (in tens of mV); in the case of a brushless, the voltage on the first coil, in the case of the only DC.

Continued on next page

Table 6.165 – continued from previous page

int16_t	WindingVoltageB	In the case step motor, the voltage across the winding B (in tens of mV); in case of a brushless, the voltage on the second winding, and in the case of DC is not used.
int16_t	WindingVoltageC	In the case of a brushless, the voltage on the third winding (in tens of mV), in the case step motor and DC is not used.
int16_t	WindingCurrentA	In the case step motor, the current in the coil A (in mA); brushless if the current in the first coil, and in the case of a single DC.
int16_t	WindingCurrentB	In the case step motor, the current in the coil B (in mA); brushless if the current in the second coil, and in the case of DC is not used.
int16_t	WindingCurrentC	In the case of a brushless, the current in the third winding (in mA), in the case step motor and DC is not used.
uint16_t	Pot	Analog input value in ten-thousandths. Range: 0..10000
uint16_t	Joy	The joystick position in the ten-thousandths. Range: 0..10000
int16_t	DutyCycle	Duty cycle of PWM.
uint8_t	Reserved [14]	Reserved (14 bytes)
uint16_t	CRC	Checksum

Description: Return device electrical parameters, useful for charts. Useful function that fill structure with snapshot of controller voltages and currents.

6.2.6.81 Command GETI

Command code (CMD): “geti” or 0x69746567.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (36 bytes)

uint32_t	CMD	Command
int8_t	Manufacturer	Manufacturer
int8_t	ManufacturerId	Manufacturer id
int8_t	ProductDescription	Product description
uint8_t	Major	The major number of the hardware version.
uint8_t	Minor	Minor number of the hardware version.
uint16_t	Release	Number of edits this release of hardware.
uint8_t	Reserved [12]	Reserved (12 bytes)
uint16_t	CRC	Checksum

Description: Return device information. It’s available from the firmware and bootloader.

6.2.6.82 Command GETM

Command code (CMD): “getm” or 0x6D746567.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (216 bytes)

uint32_t	CMD	Command
int32_t	Speed	Current speed in microsteps per second (whole steps are recalculated taking into account the current step division mode) or encoder counts per second.
int32_t	Error	Current error in microsteps per second (whole steps are recalculated taking into account the current step division mode) or encoder counts per second.
uint32_t	Length	Length of actual data in buffer.
uint8_t	Reserved [6]	Reserved (6 bytes)
uint16_t	CRC	Checksum

Description: A command to read the data buffer to build a speed graph and a sequence error. Filling the buffer starts with the command ‘start_measurements’. The buffer holds 25 points, the points are taken with a period of 1 ms. To create a robust system, read data every 20 ms, if the buffer is completely full, then it is recommended to repeat the readings every 5 ms until the buffer again becomes filled with 20 points.

6.2.6.83 Command GETS

Command code (CMD): “gets” or 0x73746567.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (54 bytes)

uint32_t	CMD	Command
uint8_t	MoveSts	Move state. This is a bit mask for bitwise operations.
	0x1 - MOVE_STATE_MOVING	This flag indicates that controller is trying to move the motor. Don’t use this flag for waiting of completion of the movement command. Use MVCMD_RUNNING flag from the MvCmdSts field instead.
	0x2 - MOVE_STATE_TARGET_SPEED	Target speed is reached, if flag set.
	0x4 - MOVE_STATE_ANTIPLAY	Motor is playing compensation, if flag set.
uint8_t	MvCmdSts	Move command state. This is a bit mask for bitwise operations.

Continued on next page

Table 6.171 – continued from previous page

	0x3f - MVCMD_NAME_BITS	Move command bit mask.
	0x0 - MVCMD_UKNWN	Unknown command.
	0x1 - MVCMD_MOVE	Command move.
	0x2 - MVCMD_MOVR	Command movr.
	0x3 - MVCMD_LEFT	Command left.
	0x4 - MVCMD_RIGHT	Command rigt.
	0x5 - MVCMD_STOP	Command stop.
	0x6 - MVCMD_HOME	Command home.
	0x7 - MVCMD_LOFT	Command loft.
	0x8 - MVCMD_SSTP	Command soft stop.
	0x40 - MVCMD_ERROR	Finish state (1 - move command have finished with an error, 0 - move command have finished correctly). This flags is actual when MVCMD_RUNNING signals movement finish.
	0x80 - MVCMD_RUNNING	Move command state (0 - move command have finished, 1 - move command is being executed).
uint8_t	PWRSts	Power state of the stepper motor (used only with stepper motor). This is a bit mask for bitwise operations.
	0x0 - PWR_STATE_UNKNOWN	Unknown state, should never happen.
	0x1 - PWR_STATE_OFF	Motor windings are disconnected from the driver.
	0x3 - PWR_STATE_NORM	Motor windings are powered by nominal current.
	0x4 - PWR_STATE_REDUCT	Motor windings are powered by reduced current to lower power consumption.
	0x5 - PWR_STATE_MAX	Motor windings are powered by maximum current driver can provide at this voltage.
uint8_t	EncSts	Encoder state. This is a bit mask for bitwise operations.
	0x0 - ENC_STATE_ABSENT	Encoder is absent.
	0x1 - ENC_STATE_UNKNOWN	Encoder state is unknown.
	0x2 - ENC_STATE_MALFUNC	Encoder is connected and malfunctioning.
	0x3 - ENC_STATE_REVERS	Encoder is connected and operational but counts in other direction.
	0x4 - ENC_STATE_OK	Encoder is connected and working properly.
uint8_t	WindSts	Windings state. This is a bit mask for bitwise operations.
	0x0 - WIND_A_STATE_ABSENT	Winding A is disconnected.
	0x1 - WIND_A_STATE_UNKNOWN	Winding A state is unknown.
	0x2 - WIND_A_STATE_MALFUNC	Winding A is short-circuited.
	0x3 - WIND_A_STATE_OK	Winding A is connected and working properly.
	0x0 - WIND_B_STATE_ABSENT	Winding B is disconnected.
	0x10 - WIND_B_STATE_UNKNOWN	Winding B state is unknown.
	0x20 - WIND_B_STATE_MALFUNC	Winding B is short-circuited.

Continued on next page

Table 6.171 – continued from previous page

	0x30 - WIND_B_STATE_OK	Winding B is connected and working properly.
int32_t	CurPosition	Current position.
int16_t	uCurPosition	Step motor shaft position in microsteps. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings). Used only with stepper motor.
int64_t	EncPosition	Current encoder position.
int32_t	CurSpeed	Motor shaft speed in steps/s or rpm.
int16_t	uCurSpeed	Part of motor shaft speed in microsteps. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings). Used only with stepper motor.
int16_t	Ipwr	Engine current, mA.
int16_t	Upwr	Power supply voltage, tens of mV.
int16_t	Iusb	USB current, mA.
int16_t	Uusb	USB voltage, tens of mV.
int16_t	CurT	Temperature in tenths of degrees C.
uint32_t	Flags	Set of flags specify motor shaft movement algorithm and list of limitations. This is a bit mask for bitwise operations.
	0x3f - STATE_CONTR	Flags of controller states.
	0x1 - STATE_ERRC	Command error encountered.
	0x2 - STATE_ERRD	Data integrity error encountered.
	0x4 - STATE_ERRV	Value error encountered.
	0x10 - STATE_EEPROM_CONNECTED	EEPROM with settings is connected.
	0x20 - STATE_IS_HOMED	Calibration performed
	0x1b3ffc0 - STATE_SECUR	Flags of security.
	0x40 - STATE_ALARM	Controller is in alarm state indicating that something dangerous had happened. Most commands are ignored in this state. To reset the flag a STOP command must be issued.
	0x80 - STATE_CTP_ERROR	Control position error(is only used with stepper motor).
	0x100 - STATE_POWER_OVERHEAT	Power driver overheat.
	0x200 - STATE_CONTROLLER_OVERHEAT	Controller overheat.
	0x400 - STATE_OVERLOAD_POWER_VOLTAGE	Power voltage exceeds safe limit.
	0x800 - STATE_OVERLOAD_POWER_CURRENT	Power current exceeds safe limit.
	0x1000 - STATE_OVERLOAD_USB_VOLTAGE	USB voltage exceeds safe limit.
	0x2000 - STATE_LOW_USB_VOLTAGE	USB voltage is insufficient for normal operation.
	0x4000 - STATE_OVERLOAD_USB_CURRENT	USB current exceeds safe limit.
	0x8000 - STATE_BORDERS_SWAP_MISSET	Engine stuck at the wrong edge.
	0x10000 - STATE_LOW_POWER_VOLTAGE	Power voltage is lower than Low Voltage Protection limit

Continued on next page

Table 6.171 – continued from previous page

	0x20000 - STATE_H_BRIDGE_FAULT	Signal from the driver that fault happened
	0x100000 - STATE_WINDING_RES_MISMATCH	The difference between winding resistances is too large
	0x200000 - STATE_ENCODER_FAULT	Signal from the encoder that fault happened
	0x800000 - STATE_ENGINE_RESPONSE_ERROR	Error response of the engine control action.
	0x1000000 - STATE_EXTIO_ALARM	The error is caused by the input signal..
uint32_t	GPIOFlags	Set of flags of gpio states. This is a bit mask for bitwise operations.
	0xffff - STATE_DIG_SIGNAL	Flags of digital signals.
	0x1 - STATE_RIGHT_EDGE	Engine stuck at the right edge.
	0x2 - STATE_LEFT_EDGE	Engine stuck at the left edge.
	0x4 - STATE_BUTTON_RIGHT	Button 'right' state (1 if pressed).
	0x8 - STATE_BUTTON_LEFT	Button 'left' state (1 if pressed).
	0x10 - STATE_GPIO_PINOUT	External GPIO works as Out, if flag set; otherwise works as In.
	0x20 - STATE_GPIO_LEVEL	State of external GPIO pin.
	0x200 - STATE_BRAKE	State of Brake pin. Flag '1' - if the pin state brake is not powered(brake is clamped), '0' - if the pin state brake is powered(brake is unclamped).
	0x400 - STATE_REV_SENSOR	State of Revolution sensor pin.
	0x800 - STATE_SYNC_INPUT	State of Sync input pin.
	0x1000 - STATE_SYNC_OUTPUT	State of Sync output pin.
	0x2000 - STATE_ENC_A	State of encoder A pin.
	0x4000 - STATE_ENC_B	State of encoder B pin.
uint8_t	CmdBufFreeSpace	This field is a service field. It shows the amount of free cells buffer synchronization chain.
uint8_t	Reserved [4]	Reserved (4 bytes)
uint16_t	CRC	Checksum

Description: Return device state. Useful function that fills structure with snapshot of controller state, including speed, position and boolean flags.

6.2.6.84 Command GFWV

Command code (CMD): "gfwv" or 0x76776667.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (10 bytes)

uint32_t	CMD	Command
uint8_t	Major	Firmware major version number
uint8_t	Minor	Firmware minor version number
uint16_t	Release	Firmware release version number

Continued on next page

Table 6.173 – continued from previous page

uint16_t	CRC	Checksum

Description: Read controller's firmware version.

6.2.6.85 Command GOFW

Command code (CMD): "gofw" or 0x77666F67.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (15 bytes)

uint32_t	CMD	Command
uint8_t	sresult	Result of command.
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Description: Command initiates the transfer of control to firmware. This command is also available from the firmware for compatibility. Result = RESULT_OK, if the transition from the loader in the firmware is possible. After the response to this command is executed transition. Result = RESULT_NO_FIRMWARE, if the firmware is not found. Result = RESULT_ALREADY_IN_FIRMWARE, if this command is called from the firmware.

6.2.6.86 Command GPOS

Command code (CMD): "gpos" or 0x736F7067.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (26 bytes)

uint32_t	CMD	Command
int32_t	Position	The position of the whole steps in the engine
int16_t	uPosition	Microstep position is only used with stepper motors. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
int64_t	EncPosition	Encoder position.
uint8_t	Reserved [6]	Reserved (6 bytes)
uint16_t	CRC	Checksum

Description: Reads the value position in steps and micro for stepper motor and encoder steps all engines.

6.2.6.87 Command GSER

Command code (CMD): “gser” or 0x72657367.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (10 bytes)

uint32_t	CMD	Command
uint32_t	SerialNumber	Board serial number.
uint16_t	CRC	Checksum

Description: Read device serial number.

6.2.6.88 Command GUID

Command code (CMD): “guid” or 0x64697567.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (40 bytes)

uint32_t	CMD	Command
uint32_t	UniqueID0	Unique ID 0.
uint32_t	UniqueID1	Unique ID 1.
uint32_t	UniqueID2	Unique ID 2.
uint32_t	UniqueID3	Unique ID 3.
uint8_t	Reserved [18]	Reserved (18 bytes)
uint16_t	CRC	Checksum

Description: This value is unique to each individual die but is not a random value. This unique device identifier can be used to initiate secure boot processes or as a serial number for USB or other end applications.

6.2.6.89 Command HASF

Command code (CMD): “hasf” or 0x66736168.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (15 bytes)

uint32_t	CMD	Command
uint8_t	sresult	Result of command.
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Description: Check for firmware on device. Result = RESULT_NO_FIRMWARE, if the firmware is not found. Result = RESULT_HAS_FIRMWARE, if the firmware found.

6.2.6.90 Command HOME

Command code (CMD): “home” or 0x656D6F68.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: The positive direction is to the right. A value of zero reverses the direction of the direction of the flag, the set speed. Restriction imposed by the trailer, act the same, except that the limit switch contact does not stop. Limit the maximum speed, acceleration and deceleration function. 1) moves the motor according to the speed FastHome, uFastHome and flag HOME_DIR_FAST until limit switch, if the flag is set HOME_STOP_ENDS, until the signal from the input synchronization if the flag HOME_STOP_SYNC (as accurately as possible is important to catch the moment of operation limit switch) or until the signal is received from the speed sensor, if the flag HOME_STOP_REV_SN 2) then moves according to the speed SlowHome, uSlowHome and flag HOME_DIR_SLOW until signal from the clock input, if the flag HOME_MV_SEC. If the flag HOME_MV_SEC reset skip this paragraph. 3) then move the motor according to the speed FastHome, uFastHome and flag HOME_DIR_SLOW a distance HomeDelta, uHomeDelta. description of flags and variable see in description for commands GHOM/SHOM

6.2.6.91 Command IRND

Command code (CMD): “irnd” or 0x646E7269.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (24 bytes)

uint32_t	CMD	Command
uint8_t	key	Random key.
uint8_t	Reserved [2]	Reserved (2 bytes)
uint16_t	CRC	Checksum

Description: Read random number from controller.

6.2.6.92 Command LEFT

Command code (CMD): “left” or 0x7466656C.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Start continous moving to the left.

6.2.6.93 Command LOFT

Command code (CMD): “loft” or 0x74666F6C.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Upon receiving the command ‘loft’ the engine is shifted from the current point to a distance GENG :: Antiplay, then move to the same point.

6.2.6.94 Command MOVE

Command code (CMD): “move” or 0x65766F6D.

Request: (18 bytes)

uint32_t	CMD	Command
int32_t	Position	Desired position (full steps or encoder counts).
int16_t	uPosition	The fractional part of a position in microsteps. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings). Is only used with stepper motor.
uint8_t	Reserved [6]	Reserved (6 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Upon receiving the command ‘move’ the engine starts to move with pre-set parameters (speed, acceleration, retention), to the point specified to the Position, uPosition. For stepper motor uPosition sets the microstep, for DC motor this field is not used.

6.2.6.95 Command MOVR

Command code (CMD): “movr” or 0x72766F6D.

Request: (18 bytes)

uint32_t	CMD	Command
int32_t	DeltaPosition	Shift (delta) of position (full steps or encoder counts)

Continued on next page

Table 6.194 – continued from previous page

int16_t	uDeltaPosition	Fractional part of the shift in micro steps is only used with stepper motor. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).
uint8_t	Reserved [6]	Reserved (6 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Move to offset. Upon receiving the command 'movr' engine starts to move with pre-set parameters (speed, acceleration, hold), left or right (depending on the sign of DeltaPosition) by the number of pulses specified in the fields DeltaPosition, uDeltaPosition. For stepper motor uDeltaPosition sets the microstep, for DC motor this field is not used.

6.2.6.96 Command PWOF

Command code (CMD): "pwof" or 0x666F7770.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Immediately power off motor regardless its state. Shouldn't be used during motion as the motor could be power on again automatically to continue movement. The command is designed for manual motor power off. When automatic power off after stop is required, use power management system.

6.2.6.97 Command RDAN

Command code (CMD): "rdan" or 0x6E616472.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (76 bytes)

uint32_t	CMD	Command
uint16_t	A1Voltage_ADC	'Voltage on pin 1 winding A' raw data from ADC.
uint16_t	A2Voltage_ADC	'Voltage on pin 2 winding A' raw data from ADC.

Continued on next page

Table 6.199 – continued from previous page

uint16_t	B1Voltage_ADC	'Voltage on pin 1 winding B' raw data from ADC.
uint16_t	B2Voltage_ADC	'Voltage on pin 2 winding B' raw data from ADC.
uint16_t	SupVoltage_ADC	'Voltage on the top of MOSFET full bridge' raw data from ADC.
uint16_t	ACurrent_ADC	'Winding A current' raw data from ADC.
uint16_t	BCurrent_ADC	'Winding B current' raw data from ADC.
uint16_t	FullCurrent_ADC	'Full current' raw data from ADC.
uint16_t	Temp_ADC	Voltage from temperature sensor, raw data from ADC.
uint16_t	Joy_ADC	Joystick raw data from ADC.
uint16_t	Pot_ADC	Voltage on analog input, raw data from ADC
uint16_t	L5_ADC	USB supply voltage after the current sense resistor, from ADC.
uint16_t	H5_ADC	Power supply USB from ADC
int16_t	A1Voltage	'Voltage on pin 1 winding A' calibrated data (in tens of mV).
int16_t	A2Voltage	'Voltage on pin 2 winding A' calibrated data (in tens of mV).
int16_t	B1Voltage	'Voltage on pin 1 winding B' calibrated data (in tens of mV).
int16_t	B2Voltage	'Voltage on pin 2 winding B' calibrated data (in tens of mV).
int16_t	SupVoltage	'Voltage on the top of MOSFET full bridge' calibrated data (in tens of mV).
int16_t	ACurrent	'Winding A current' calibrated data (in mA).
int16_t	BCurrent	'Winding B current' calibrated data (in mA).
int16_t	FullCurrent	'Full current' calibrated data (in mA).
int16_t	Temp	Temperature, calibrated data (in tenths of degrees Celcius).
int16_t	Joy	Joystick, calibrated data. Range: 0..10000
int16_t	Pot	Analog input, calibrated data. Range: 0..10000
int16_t	L5	USB supply voltage after the current sense resistor (in tens of mV).
int16_t	H5	Power supply USB (in tens of mV).
uint16_t	deprecated	
int32_t	R	Motor winding resistance in mOhms(is only used with stepper motor).
int32_t	L	Motor winding pseudo inductance in uHn(is only used with stepper motor).
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Description: Read analog data structure that contains raw analog data from ADC embedded on board. This function used for device testing and deep recalibraton by manufacturer only.

6.2.6.98 Command READ

Command code (CMD): “read” or 0x64616572.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Read all settings from controller’s flash memory to controller’s RAM, replacing previous data in controller’s RAM.

6.2.6.99 Command RERS

Command code (CMD): “rers” or 0x73726572.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Read important settings (calibration coefficients and etc.) from controller’s flash memory to controller’s RAM, replacing previous data in controller’s RAM.

6.2.6.100 Command REST

Command code (CMD): “rest” or 0x74736572.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: The reset command controller and go into bootloader mode, for compatibility with the exchange protocol to the loader. No response to this command.

6.2.6.101 Command RIGT

Command code (CMD): “rigt” or 0x74676972.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Start continous moving to the right.

6.2.6.102 Command SARS

Command code (CMD): "sars" or 0x73726173.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Save important settings (calibration coefficients and etc.) from controller's RAM to controller's flash memory, replacing previous data in controller's flash memory.

6.2.6.103 Command SAVE

Command code (CMD): "save" or 0x65766173.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Save all settings from controller's RAM to controller's flash memory, replacing previous data in controller's flash memory.

6.2.6.104 Command SPOS

Command code (CMD): "spos" or 0x736F7073.

Request: (26 bytes)

uint32_t	CMD	Command
int32_t	Position	The position of the whole steps in the engine
int16_t	uPosition	Microstep position is only used with stepper motors. Microstep size and the range of valid values for this field depend on selected step division mode (see MicrostepMode field in engine_settings).

Continued on next page

Table 6.212 – continued from previous page

int64_t	EncPosition	Encoder position.
uint8_t	PosFlags	Flags. This is a bit mask for bitwise operations.
	0x1 - SETPOS_IGNORE_POSITION	Will not reload position in steps/microsteps if this flag is set.
	0x2 - SETPOS_IGNORE_ENCODER	Will not reload encoder state if this flag is set.
uint8_t	Reserved [5]	Reserved (5 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Sets any position value in steps and micro for stepper motor and encoder steps of all engines. It means, that changing main indicator of position.

6.2.6.105 Command SSER

Command code (CMD): "sser" or 0x72657373.

Request: (50 bytes)

uint32_t	CMD	Command
uint32_t	SN	New board serial number.
uint8_t	Key	Protection key (256 bit).
uint8_t	Major	The major number of the hardware version.
uint8_t	Minor	Minor number of the hardware version.
uint16_t	Release	Number of edits this release of hardware.
uint8_t	Reserved [4]	Reserved (4 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Write device serial number and hardware version to controller's flash memory. Along with the new serial number and hardware version a 'Key' is transmitted. The SN and hardware version are changed and saved when keys match. Can be used by manufacturer only.

6.2.6.106 Command SSTP

Command code (CMD): "sstp" or 0x70747373.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Soft stop engine. The motor stops with deceleration speed.

6.2.6.107 Command STMS

Command code (CMD): “stms” or 0x736D7473.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Start measurements and buffering of speed, following error.

6.2.6.108 Command STOP

Command code (CMD): “stop” or 0x706F7473.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Immediately stop the engine, the transition to the STOP, mode key BREAK (winding short-circuited), the regime ‘retention’ is deactivated for DC motors, keeping current in the windings for stepper motors (with Power management settings). When this command is called, the ALARM flag is reset.

6.2.6.109 Command UPDF

Command code (CMD): “updf” or 0x66647075.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Command puts the controller to update the firmware. After receiving this command, the firmware board sets a flag (for loader), sends echo reply and restarts the controller.

6.2.6.110 Command WDAT

Command code (CMD): “wdat” or 0x74616477.

Request: (142 bytes)

uint32_t	CMD	Command
uint8_t	Data	Encoded firmware.
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Writes encoded firmware in the controller flash memory. Result of each packet write is not available. Overall result is available when firmware upload is finished.

6.2.6.111 Command WKEY

Command code (CMD): “wkey” or 0x79656B77.

Request: (46 bytes)

uint32_t	CMD	Command
uint8_t	Key	Protection key (256 bit).
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Answer: (15 bytes)

uint32_t	CMD	Command
uint8_t	sresult	Result of command.
uint8_t	Reserved [8]	Reserved (8 bytes)
uint16_t	CRC	Checksum

Description: Write command key to decrypt the firmware. Result = RESULT_OK, if the command loader. Result = RESULT_HARD_ERROR, if at the time of the command there was mistake. Result not available using the library write_key, the field value is processed by the function. Can be used by manufacturer only.

6.2.6.112 Command ZERO

Command code (CMD): “zero” or 0x6F72657A.

Request: (4 bytes)

uint32_t	CMD	Command

Answer: (4 bytes)

uint32_t	CMD	Command

Description: Sets the current position and the position in which the traffic moves by the move command and movr zero for all cases, except for movement to the target position. In the latter case, set the zero current position and the target position counted so that the absolute position of the destination is the same. That is, if we were at 400 and moved to 500, then the command Zero makes the current position of 0, and the position of the destination - 100. Does not change the mode of movement that is if the motion is carried, it continues, and if the engine is in the 'hold', the type of retention remains.

About this document

6.3 8SMC1-USBhF software compatibility

Important: The communication protocols for 8SMC1 and 8SMC4/5 controllers are not compatible. The communication protocol for 8SMC4/5 is described in the *Communication protocol specification*. Due to protocol differences certain amount of functionality of both 8SMC1 and 8SMC4/5 cannot be mapped onto each other. The recommended way to work with 8SMC4/5 would be to adapt your software to use libximc library.

XiLab software is designed for 8SMC4 and 8SMC5 controllers, **it won't work with 8SMC1**. For 8SMC1 there are two software packs: MicroSMC and SMCVieW.

- SMCView is a Labview-based GUI for 8SMC1 controllers. It uses one kind of driver to work with 8SMC1.
- MicroSMC is a program-driver for 8SMC1 controllers. It uses another kind of driver to work with 8SMC1 (not the one which SMCView uses).

New motor controllers can be used with the software written for the 8SMC1-USBhF series. There are two options available to control new motor controllers using 8SMC1-USBhF-compatible software:

1. Recommended. Use MicroSMC software with support for both 8SMC1-USBhF and new motor controllers. In this case MicroSMC process will open all found motor controllers in exclusive mode to arbitrate access and it will be possible to control them through USMCDLL, which uses MicroSMC. Download MicroSMC with support for new motor controllers on the *Software compatibility* page.
2. Use USMCDLL/libximc compatibility library to control motor controllers without MicroSMC software. Use this option if you need to simultaneously run MicroSMC and work with some motor controllers using libximc library. Download USMCDLL/libximc compatibility library here.

When move from the 8SMC1 controller to the 8SMC4/5 we advice to replace your program functions to use the libximc library. The table below shows the mapping from 8SMC1-USBhF to new motor controller functions: first column lists USMCDLL library functions, second column lists corresponding paragraph in 8SMC1-USBhF controller user's manual, third column contains this function features in the migration library.

Function call	8SMC1-USBhF	New motor controller
USMC_Init	7.5.3	Uses libximc *enumerate_devices*, *get_enumerate_device_information*, *get_enumerate_device_serial*, *get_device_count*, *get_device_name*, *open_device* functions. Queries all COM-ports present in the system.
USMC_GetState	7.5.4	Uses libximc *get_status* and *get_engine_settings* functions. AReset, EMReset, RotTrErr flags in USMC_State structure are always set to false.
USMC_SaveParametersToFlash	7.5.5	Uses libximc *command_save_settings* function.

Continued on next page

Table 6.230 – continued from previous page

Function call	8SMC1-USBhF	New motor controller
USMC_GetMode	7.5.6	Uses libximc *get_edges_settings*, *get_power_settings*, *get_control_settings*, *get_ctp_settings*, *get_sync_out_settings* functions. EMReset, ResetRT, SyncOUTR, EncoderEn, EncoderInv, ResBEnc, ResEnc flags in USMC_Mode are always set to false, SyncINOp flag is always set to true.
USMC_SetMode	7.5.7	Uses all functions used by USMC_GetMode and their "set_" equivalents. Ignores RotTeEn, RotTrOp, ResetRT, SyncOUTR, SyncINOp, EncoderEn flags.
USMC_GetParameters	7.5.8	Uses libximc *get_secure_settings*, *get_engine_settings*, *get_move_settings*, *get_feedback_settings*, *get_power_settings*, *get_control_settings*, *get_ctp_settings*, *get_home_settings*, *get_sync_out_settings* functions. Returns zeroes in BTimeoutR, BTimeoutD, MinP, MaxLoft, StartPos fields.
USMC_SetParameters	7.5.9	Uses all functions used by USMC_GetParameters and their "set_" equivalents. Ignores BTimeoutR, BTimeoutD, MinP, MaxLoft, StartPos parameters.
USMC_GetStartParameters	7.5.10	Uses libximc *get_move_settings*, *get_engine_settings* functions. WSyncIN, SyncOUTR, ForceLoft flags in USMC_StartParameters structure are always set to false.
USMC_Start	7.5.11	Uses libximc *get_move_settings*, *get_engine_settings*, *set_move_settings*, *set_engine_settings*, *command_move* functions.
USMC_Stop	7.5.12	Uses libximc *command_stop* function.
USMC_SetCurrentPosition	7.5.13	Uses libximc *set_position* function.
USMC_GetEncoderState	7.5.14	Uses libximc *get_status* function.
USMC_GetLastErr	7.5.15	Does not modify its arguments. Error status is indicated by the nonzero return code of the corresponding USMC_ function.
USMC_Close	7.5.16	Uses libximc *close_device* function.

Migration library does not require and does not interact with background MicroSMC.exe process. It uses libximc.dll library to interface with new motor controllers.

All composite USMC_ functions containing several libximc function calls terminate if any of the underlying libximc functions returns an error. In this case controller may be left with partially saved settings. Nonzero return code of USMC_ functions is equal to the return code of the libximc function which returned an error.

Application example: usmcdll_libximc_test.zip

To make 8SMC1-USBhF work on Windows 10 you would first need to update controller firmware to the latest version (25.05). All relevant software can be found on this page. Full Software & Support Package for Windows 8. Latest Firmware v.2505 - updated for Windows 8 and 10.

Important: To update the firmware you will need Windows 7 or Windows 8.1. Most likely you will not be able to update the firmware running Windows 10

If you want to repair the 8SMC1-USBhF controllers, please contact the manager who sold the equipment. If the manager's contacts was lost, you can contact sales@standa.lt. **Important**, the cost of repair can be comparable to the

cost of a new controller.

6.4 Libximc library timeouts

A number of timeouts and wait times are used when working with *XiLab* program or your own application using *libximc library* to detect errors and support robust controller operation. A list of times is provided below, together with reasons. Times are optimized for the USB connection on a modern PC. It is important to take delays into account when designing your own signal transmission lines to avoid erroneous timeouts.

Condition	Name	Time in milliseconds
Enumeration timeout. Happens if device type cannot be determined.	ENUMERATE_TIMEOUT_TIME	100
Port open timeout. Happens if library cannot open port.	DEFAULT_TIMEOUT_TIME	5000
Wait time when no data is received from device.	DEFAULT_TIMEOUT_TIME	5000
Wait time on device open.	RESET_TIME/2	50
Wait time from controller reset to device reappearance on firmware reflash.	RESET_TIME * 1.2 + DEFAULT_TIMEOUT_TIME	5120
Wait time on flash sector write.	FLASH_SECTIONWRITE_TIME	100
Reconnect timeout on flash update .	XISM_PORT_DETECT_TIME	60000

6.5 XILab scripts

- *Brief description of the language*
  - *Data Types*
  - *Statements*
  - *Variable statements*
  - *Reserved words*
  - *Functions*
- *Syntax highlighting*
- *Additional XILab functions*
  - *XILab log*
  - *Script execution delay*
  - *New axis object creation*
  - *New file object creation*
  - *Creation of calibration structure*
  - *Get next serial*
  - *Wait for stop*
  - *libximc library functions*
- *Examples*

- *Bit mask example script*
- *A script which scans and writes data to the file*
- *Multi axis cyclic movement script*
- *Single axis cyclic movement script*
- *Homing test script*
- *List axis serials script*
- *Move and wait script*
- *Random shift script*
- *Set zero scrip*
- *Autotester script*
- *Border crossing test*
- *Closed loop tuning test*
- *Discrete motion script*
- *Exponential position change in user units script*
- *For calb step script*
- *Step script*
- *Homing test with extio*
- *Motion by sin function*
- *Move EXTIO calb script*
- *Probabilistic tests*
- *Several shifts with calibration script*
- *Steps loss test*
- *Sync test script*
- *Sync bug test script*

XILab scripting language is implemented using QtScript, which in turn is based on ECMAScript.

ECMAScript is the scripting language standardized by Ecma International in the ECMA-262 specification and ISO/IEC 16262.

QtScript (and, by extension, XILab) uses third edition of the ECMAScript standard .

6.5.1 Brief description of the language

6.5.1.1 Data Types

ECMAScript supports nine primitive data types. Values of type Reference, List, and Completion are used only as intermediate results of expression evaluation and cannot be stored as properties of objects. The rest of the types are:

- Undefined,
- Null,
- Boolean,

- String,
- Number,
- Object.

6.5.1.2 Statements

Most common ECMAScript language statements are summarized below:

Name	Usage	Description
Block	{[<statement list>]}	Several statements may be grouped into a block using braces.
Variable declaration	var <varialble declaration list>	Variables are declared using "var" keyword.
Empty statement	;	Semicolon denotes an empty instruction. It is not required to end a line with a semicolon.
Conditional execution	if (<condition>) <instruction> [else <instruction>]	Conditional execution is done using "if ... else" keywords. If a condition is true, then "if"-block instruction is executed, else an "else"-block instruction is executed.
Loop	do <loop body> while (<condition>) while (<condition>) <loop body> for ([<initialization>]; [<condition>]; [<iterative statement>]) <loop body>	Loops have several forms. A "do ... while ..." loop executes loop body and then checks if condition is true or false to see whether it should stop or continue running. A "while ... do ..." loop repeatedly checks the condition and executes loop body if it is true. A "for ..." loop executes an initialization statement once, then executes an iterative statement and loop body while the condition is true.
Return	return [<expression>]	Stops function execution and returns expression as a result.
Exception	throw <expression>	Generates or "throws" an exception, which may be processed by the "try" statement (see below).
Try-catch block	try <block> catch (<identifier>) <block> try <block> finally <block> try <block> catch (<identifier>) <block> finally <block>	Used together with exceptions. This statement tries to execute its "try"-block. If an exception is thrown in it, then a "catch"-block is executed. Finally a "finally"-block is executed unconditionally. Either a "catch" or a "finally" block may be omitted.

6.5.1.3 Variable statements

Variables are declared using *var* keyword. A declared variable is placed within visibility scope that corresponds to the function in which it is declared. If the variable is declared outside of functions, it is placed in the global visibility scope. Variable is created when the function within which it was declared, or, if the variable is global, at the start of the application. When a variable is created it is initialized with *Undefined* value. If a variable is created with initialization, the initialization does not occur in the moment of variable creation, it happens when the string with the var statement executes.

6.5.1.4 Reserved words

The following words are the reserved keywords in the language and may not be used as identifiers:

```
break     else       new     var
case      finally    return  void
catch     for        switch  while
continue  function   this    with
default   if         throw
delete    in         try
do        instanceof typeof
```

The following words are used as keywords in proposed extensions and are therefore reserved to allow for the possibility of future adoption of those extensions:

```
abstract  enum       int        short
boolean   export     interface  static
byte      extends    long       super
char      final      native     synchronized
class     float      package    throws
const     goto       private    transient
debugger  implements protected  volatile
double    import     public
```

6.5.1.5 Functions

Functions are objects in ECMAScript. Functions like any other objects can be stored in variables, objects and arrays, can be passed as arguments to other functions and can be returned by functions. Functions, like any other objects may have properties. Essential specific feature of functions is that they can be invoked.

In the application text, the most common way to define a function is:

```
function sum(arg1, arg2) { // a function which takes two parameters
    return arg1 + arg2;    // and returns their sum
}
```

6.5.2 Syntax highlighting

Script window text has syntax highlighting. Its colors are:

Statement type	*color*	*text example*
Arbitrary functions	purple	my_function();
XILab functions	blue	get_status();
Positive numbers	green	a = 100;
Negative numbers	red	b = -200;
Comments	grey	// a comment
The rest of the text	black	var s = "a string";

During the script execution the background of line with the last executed command is changed to dark gray with update rate of once in every 20 ms.

6.5.3 Additional XILab functions

This image shows XILab functions which are available from scripts, aside from standard built-in language functions.

var s = "a string";

- log(string text [, int loglevel]) – save text to the XILab log
- msleep(int ms) - delay script execution

- new_axis(int serial_number) - create new axis object
- new_file(string filename) - create new file object
- new_calibration(int A, int Microstep) - create calibration structure to pass to calibrated functions
- get_next_serial(int serial) - get next serial out of an ordered list of opened controller serials
- command_wait_for_stop(int refresh_period) - wait until controller stops moving
- and all libximc library functions (see *Programming guide*)

Also, all constant values from the communication protocol are defined and can be used in scripts. *Usage example*.

6.5.3.1 XILab log

Logging is done by calling log(string text [, int loglevel]) function. This function adds the *text* line to the XILab log. If the second *loglevel* parameter is passed the message receives the appropriate logging level and is displayed in corresponding color.

Loglevel	Type
1	Error
2	Warning
3	Info

Example:

```
var x = 5;
log("x = " + x);
```

Function usage example

Note: It is not recommended to invoke functions that interact with XILab user interface (i.e. logging function) with a frequency of more than once in 20 ms.

6.5.3.2 Script execution delay

Script is paused by calling the msleep(int ms) function, which suspends script execution for *ms* milliseconds.

Example:

```
msleep(200);
```

Function usage example.

6.5.3.3 New axis object creation

XILab multi-axis interface provides the ability to manage controllers via scripts. The difference from the single-axis case is that you should specify the controller which receives the command. An "axis" object is introduced to abstract this concept. It has methods which match the *libximc library* function names. Controllers are identified by their serial numbers.

Example:

```
var x = new_axis(123);
x.command_move(50);
```

In this example first line of the script creates an axis-type object with the variable name "x", which tries to use controller with the serial number "123". If this controller is not connected, then the script will return an error and terminate. The second line of the script sends a "move to position 50" command to this controller.

Function usage example.

6.5.3.4 New file object creation

XILab scripts can read from and write to files. To do this you need to create a "file" object, passing desired filename in its constructor. File object has the following functions:

return_type Function_name	Description
bool open()	Opens the file. File is opened in read-write mode if possible, in read-only mode otherwise.
void close()	Closes the file.
Number size()	Returns file size in bytes.
bool seek(Number pos)	Sets current position in file to *pos* bytes[1].
bool resize(Number size)	Resizes the file to *size* bytes. If *size* is less than current file size, then the file is truncated, if it is greater than current file size, then the file is padded with zero bytes.
bool remove()	Removes the file.
String read(Number maxsize)	Reads up to *maxsize* bytes from the file and returns result as a string. Data is read in utf-8 Unicode encoding.
Number write(String s, Number maxsize)	Writes up to *maxsize* btyes to the file from the string. Data is written in utf-8 unicode encoding, end-of-line character should be set by user. Returns amount of written bytes or -1 if an error occurred.

All file functions which return *bool* type, return "true" on success and "false" on failure.

Use "/" symbol as path separator, this works on all systems (Windows/Linux/Mac).

Example:

```
var winf = new_file("C:/file.txt"); // An example of file name and path on Windows
var linf = new_file("/home/user/Desktop/file.txt"); // An example of file name and␣
↪path on Linux
var macf = new_file("/Users/macuser/file.txt"); // An example of file name and path␣
↪on Mac

var f = winf; // Pick a file name
if (f.open()) { // Try to open the file
  f.write( "some text" ); // If successful, then write desired data to the file
  f.close(); // Close the file
} else { // If file open failed for some reason
  log( "Failed opening file" ); // Log an error
}
```

Function usage example.

6.5.3.5 Creation of calibration structure

new_calibration(double A, int Microstep) function takes as a parameter a floating point number A, which sets the ratio of user units to motor steps, and microstep division mode, which was either read earlier from MicrostepMode field of get_engine_settings() return type, or set by a *MICROSTEP_MODE_* constant. This function returns calibration_t

[1] Seeking beyond the end of a file: If the position is beyond the end of a file, then seek() shall not immediately extend the file. If a write is performed at this position, then the file shall be extended. The content of the file between the previous end of file and the newly written data is UNDEFINED and varies between platforms and file systems.

structure, which should be passed to calibrated get_/set* functions to get or set values in user units. The following two forms are functionally equivalent:

```
// create calibration: type 1
var calb = new_calibration(c1, c2);
```

```
// create calibration: type 2
var calb = new Object();
calb.A = c1;
calb.MicrostepMode = c2;
```

Function usage example.

6.5.3.6 Get next serial

get_next_serial(int serial) function takes as a parameter an integer number and returns the smallest serial from a sorted list of opened controller serials which is strictly greater than the parameter. If there are no such serials a zero is returned. This function is a convenient shortcut for automatic creation of "axis" type objects without hardcoded serial numbers.

Example:

```
var first_serial = get_next_serial(0);
var x = new_axis(first_serial);
var y = new_axis(get_next_serial(first_serial));
```

In this example in the first line we obtain a serial, in the second line an axis-type object is created, in the third line we get the next serial and create an axis for it.

Function usage example.

6.5.3.7 Wait for stop

The command_wait_for_stop(int refresh period) script function waits until the controller stops movement, that is, until the MVCMD_RUNNING bit in the MvCmdSts member of the structure returned by the get_status() function becomes unset. command_wait_for_stop script function uses command_wait_for_stop libximc function and takes as a paramater an integer denoting time delay in milliseconds between successive queries of controller state.

This function is also present as a method of an "axis"-type object.

Function usage example.

6.5.3.8 libximc library functions

Libximc library functions with "get*" prefix read settings from the controller and return the corresponding settings structure. Libximc library functions with "set*" prefix take as a parameter a settings data structure and write these settings to the controller. There are two ways to set data structure contents:

1. call the corresponding get-function and modify required fields

```
// set settings: type 1
var m = get_move_settings();
m.Speed = 100;
set_move_settings(m);
```

2. create an *Object* and set all of its *properties* that are present as members of the data structure (case-sensitive).

```
// set settings: type 2
var m = new Object;
m.Speed = 100;
m.uSpeed = 0;
m.Accel = 300;
m.Decel = 500;
m.AntiplaySpeed = 10;
m.uAntiplaySpeed = 0;
set_move_settings(m);
```

Please note, that in the first case controller receives an additional command (sent by the get-function before the set-). In the second case one should initialize all object properties corresponding to structure members. Any missing property will be initialized with zero. Any property that does not match a structure member name will be ignored. Any property with non-matching type will be typecast according to EcmaScript rules. All data structures are described in *Communication protocol specification* chapter of the manual.

Function usage example.

6.5.4 Examples

This section contains examples of typical tasks which can be easily automated by XILab scripts.

6.5.4.1 Bit mask example script

```
/*
* Bit mask example script
*
* Description of the script:
*  This script clearly shows how to work with bit masks.
*  This script or part of it may be needed when working with any of our other␣
↪commands that use bit masks. For example, the  «set_home_settings» command
*
* To run the script, upload it to the XILab software
*/

var a = new_axis(get_next_serial(0)); // take first found axis
var gets = a.get_status(); // read status once and reuse it

var gpio = gets.GPIOFlags;
var left = STATE_LEFT_EDGE;
var right = STATE_RIGHT_EDGE;
var mask = left | right;
var result = gpio & mask;
log( to_binary(left) + " = left limit switch flag" );
log( to_binary(right) + " = right limit switch flag" );
log( to_binary(mask) + " = OR operation on flags gives the mask" );
log( to_binary(gpio) + " = gpio state" );
log( to_binary(result) + " = AND operation on state and mask gives result" );
if ( result ) {
  log("At least one limit switch is on");
} else {
  log("Both limit switches are off");
}

// Binary representation function
function to_binary(i)
{
```

```
  bits = 32;
  x = i >>> 0; // coerce to unsigned in case we need to print negative ints
  str = x.toString(2); // the binary representation string
  return (repeat("0", bits) + str).slice (-bits); // pad with zeroes and return
}

// String repeat function
function repeat(str, times)
{
  var result="";
  var pattern=str;
  while (times > 0) {
    if (times&1) {
      result+=pattern;
    }
    times>>=1;
    pattern+=pattern;
  }
  return result;
}
```

6.5.4.2 A script which scans and writes data to the file

```
/*
* A script which scans and writes data to the file
*
* Description of the script:
*  This script scans and writes the data to a .csv file.
*  The script can be useful if you are using the system to scan an area and/or␣
↪capture frames
*
* To run the script, upload it to the XILab software
*/

var start = 0; // Starting coordinate in steps
var step = 10; // Shift amount in steps
var end = 100; // Ending coordinate in steps

var speed = 300; // maximum movement speed in steps / second
var accel = 100; // acceleration value in steps / second^2
var decel = 100; // deceleration value in steps / second^2
var delay = 100;

var m = get_move_settings(); // read movement settings from the controller
m.Speed = speed; // set movement speed
m.Accel = accel; // set acceleration
m.Decel = decel; // set deceleration
set_move_settings(m); // write movement settings into the controller

var f = new_file("C:/a.csv"); // Choose a file name and path
f.open(); // Open a file
f.seek( 0 ); // Seek to the beginning of the file

command_move(start); // Move to the starting position
command_wait_for_stop(delay); // Wait until controller stops moving

while (get_status().CurPosition < end) {
```

```
  f.write( get_status().CurPosition + "," + get_chart_data().Pot + "," + Date.now() +
↪"\n" ); // Get current position, potentiometer value and date and write them to file
  command_movr(step); // Move to the next position
  command_wait_for_stop(delay); // Wait until controller stops moving
}
f.close(); // Close the file
```

move_and_sleep.csv

- a sample file for use with the above example

6.5.4.3 Multi axis cyclic movement script

```
/*
* Multi axis cyclic movement script
*
* Description of the script:
*  Does cyclic movement between two border points with set values of acceleration,
*  deceleration and top speed, for all axes found. The script is similar to the
↪"Cyclic"
*  button in XILab
*
* To run the script, upload it to the XILab software
*/

var axes = [];
var number_of_axes = 0;
var last_serial = 0;

while (serial = get_next_serial(last_serial))   // Get next serial number and repeat␣
↪for each axes
{
  axes[number_of_axes] = new_axis(serial);
  log("Found axis " + number_of_axes + " with serial number " + serial);
  number_of_axes++;
  last_serial = serial;
}

for (var i = 0; i < number_of_axes; i++)
{
  axis_configure(axes[i]);
}

while (1)
{
  for (var i = 0; i < number_of_axes; i++)
  {
    go_first_border(axes[i]);
    go_second_border(axes[i]);
  }

  msleep(100);
}

function axis_configure(axis)
{
  var speed = 1000;   // Maximum movement speed in steps / second
  var accel = 2000;   // Acceleration value in steps / second^2
```

```
  var decel = 5000;   // Deceleration value in steps / second^2

  axis.command_stop(); // send STOP command (does immediate stop)
  axis.command_zero(); // send ZERO command (sets current position and encoder value␣
↪to zero)
  var m = axis.get_move_settings(); // read movement settings from the controller
  m.Speed = speed; // set movement speed
  m.Accel = accel; // set acceleration
  m.Decel = decel; // set deceleration
  axis.set_move_settings(m); // write movement settings into the controller
}

function go_first_border(axis)
{
  var first_border = 0; // first border coordinate in steps
  var GETS = axis.get_status();

  if (!(GETS.MvCmdSts & MVCMD_RUNNING) && (GETS.CurPosition != first_border))
  {
    axis.command_move(first_border);  // move towards one border
  }
}

function go_second_border(axis)
{
  var second_border = 25000; // second border coordinate in steps
  var GETS = axis.get_status();

  if (!(GETS.MvCmdSts & MVCMD_RUNNING) && (GETS.CurPosition != second_border))
  {
    axis.command_move(second_border);   // move towards another border
  }
}
```

6.5.4.4 Single axis cyclic movement script

```
/*
* Single axis cyclic movement script
*
* Description of the script:
*  Does cyclic movement between two border points with set values of acceleration,
*  deceleration and top speed. The script is similar to the "Cyclic" button in XILab
*
* To run the script, upload it to the XILab software
*/

var first_border = -10; // first border coordinate in mm
var second_border = 10; // second border coordinate in mm
var mm_per_step = 0.005; // steps to distance translation coefficient
var delay = 100; // delay in milliseconds
var calb = new_calibration(mm_per_step, get_engine_settings().MicrostepMode); //␣
↪create calibration structure
command_stop(); // send STOP command (does immediate stop)
command_zero(); // send ZERO command (sets current position and encoder value to zero)
while (1) { // infinite loop
  command_move_calb(first_border, calb); // move towards one border
  command_wait_for_stop(delay); // wait until controller stops moving
```

```
  command_move_calb(second_border, calb); // move towards another border
  command_wait_for_stop(delay); // wait until controller stops moving
}
```

6.5.4.5 Homing test script

```
/*
* Homing test script
*
* Description of the script:
*  This script tests homing function by repeatedly moving to a random position,
*  doing quick stop and then homing, for all axes found. The script is similar to the␣
↪GO
*  Home button in XILab
*
* To run the script, upload it to the XILab software
*/

var axes = [];
var number_of_axes = 0;
var last_serial = 0;
while (serial = get_next_serial(last_serial)) // get next serial number and repeat␣
↪for each axes.
{
    axes[number_of_axes] = new_axis(serial);
   log("Found axis " + number_of_axes + " with serial number " + serial);
   number_of_axes++;
   last_serial = serial;
}

while (1) { // infinite loop
    for (var i = 0; i < number_of_axes; i++)
   {
        homing_test(axes[i]);
    }
}

function homing_test(axis)
{
    var shift_low = 0; // minimum shift distance in steps
   var shift_high = 10000; // maximum shift distance in steps
   var speed_low = 100; // minimum movement speed in steps / second
   var speed_high = 5000; // maximum movement speed in steps / second
   var time_low = 1; // minimum wait time in seconds
   var time_high = 10; // maximum wait time in seconds

    axis.command_home(); // send HOME command (find home position)
    axis.command_wait_for_stop(100); // wait until controller stops moving
    var m = axis.get_move_settings(); // read movement settings from the controller
    m.Speed = rnd(speed_low, speed_high); // set random speed from a range of speeds␣
↪between "speed_low" and "speed_high"
    axis.set_move_settings(m); // write movement settings into the controller
    var shift = rnd(shift_low, shift_high); // pick random shift value from a range␣
↪of distances between "shift_low" and "shift_high"
    if (Math.random() < 0.5) { // pick random direction
       shift = -shift;
    }
```

```
    axis.command_movr(shift); // send MOVR command (does a relative shift)
    msleep( rnd(time_low*1000, time_high*1000) ); // pause for a random time from a␣
↪range between "time_low" and "time_high"
    axis.command_stop(); // send STOP command (does immediate stop)
}

function rnd(min,max) { // "rnd" is a helper function which uses Math.random() and␣
↪returns a uniformly distributed integer random value between "min" and "max"
 var r = Math.random()*(max-min)+min;
 return Math.round(r);
}
```

6.5.4.6 List axis serials script

```
/*
* List axis serials script
*
* Description of the script:
*  An example of a script that searches for all the serial numbers of controllers
*  and outputs them to the log.
*
* To run the script, upload it to the XILab software
*/

var i = 0; // Declare loop iteration variable
var serial = 0; // Declare serial number variable
var axes = Array(); // Declare axes array
while (true) { // The loop
 serial = get_next_serial(serial); // Get next serial
 if (serial == 0) // If there are no more controllers then...
     break; // ...break out of the loop
 var a = new Object(); // Create an object
 a.serial = serial; // Assign serial number to its "serial" property
 a.handle = new_axis(serial); // Assign new axis object to its "handle" property
 axes[i] = a; // Add it to the array
 i++; // Increment counter
}
for (var k=0; k < axes.length; k++) { // Iterate through array elements
  log ( "Axis with S/N " + axes[k].serial + " is in position " + axes[k].handle.get_
↪status().CurPosition ); // For each element print saved axis serial and call a get_
↪status() function
}
```

6.5.4.7 Move and wait script

```
/*
* Move and wait script
*
* Description of the script:
*  The script reads the next coordinate and the delay time from the csv file,
*  after which it moves to the specified coordinate with a subsequent delay,
*  and so on until the end of reading the entire file.
*     The script can be useful if you are using the system to scan an area and/or␣
↪capture frames
* To run the script, upload it to the XILab software
*/
```

```
var axis = new_axis(get_next_serial(0)); // Use first available controller
var x; // A helper variable, represents coordinate
var ms; // A helper variable, represents wait time in milliseconds
var f = new_file("./move_and_sleep.csv"); // Choose a file name and path; this script␣
↪uses a file from examples in the installation directory
f.open(); // Open a file
while ( str = f.read(4096) ) { // Read file contents string by string, assuming each␣
↪string is less than 4 KiB long
  var ar = str.split(","); // Split the string into substrings with comma as a␣
↪separator; the result is an array of strings
  x = ar[0]; // Variable assignment
  ms = ar[1]; // Variable assignment
  log( "Moving to coordinate " + x ); // Log the event
  axis.command_move(x); // Move to the position
  axis.command_wait_for_stop(100); // Wait until the movement is complete
  log( "Waiting for " + ms + " ms" ); // Log the event
  msleep(ms); // Wait for the specified amount of time
}
log ( "The end." );
f.close(); // Close the file
```

6.5.4.8 Random shift script

```
/*
* Random shift script
*
* Description of the script:
*  This script does shifts on random offset from a specified range of distances
*  with a random speed from a chosen range of speeds.

* To run the script, upload it to the XILab software
*/

var axes = [];
var number_of_axes = 0;
var last_serial = 0;
while (serial = get_next_serial(last_serial)) // get next serial number and repeat␣
↪for each axes.
{
   axes[number_of_axes] = new_axis(serial);
   log("Found axis " + number_of_axes + " with serial number " + serial);
   number_of_axes++;
    last_serial = serial;
}

while (1) { // infinite loop
   for (var i = 0; i < number_of_axes; i++)
   {
       go_to_random_shift(axes[i]);
    }
}

function go_to_random_shift(axis)
{
   var shift_low = 0; // minimum shift distance in steps
   var shift_high = 10000; // maximum shift distance in steps
   var speed_low = 100; // minimum movement speed in steps / second
```

```
   var speed_high = 5000; // maximum movement speed in steps / second

   var m = axis.get_move_settings(); // read movement settings from the controller
   m.Speed = rnd(speed_low, speed_high); // set random speed from a range of speeds␣
↪between "speed_low" and "speed_high"
   axis.set_move_settings(m); // write movement settings into the controller
   var shift = rnd(shift_low, shift_high); // pick random shift value from a range of␣
↪distances between "shift_low" and "shift_high"
   if (Math.random() < 0.5) { // pick random direction
       shift = -shift;
   }
   axis.command_movr(shift); // send MOVR command (does a relative shift)
    axis.command_wait_for_stop(100); // wait until controller stops moving
}

function rnd(min,max) { // "rnd" is a helper function which uses Math.random() and␣
↪returns a uniformly distributed integer random value between "min" and "max"
  var r = Math.random()*(max-min)+min;
  return Math.round(r);
}
```

6.5.4.9 Set zero scrip

```
/*
* Set zero script
*
* Description of the script:
*  This script changes "standoff" setting (found on "Home position" page in the XILab
↪"Settings") so that the current position becomes the home position.
*  The script is very convenient for calibrating and configuring new stages, as well␣
↪as for changing the zero position
*
* How to use:
*  - manually move your positioner to a desired position
*  - launch this script and wait for completion
*  As a result your positioner will return to the starting position and all␣
↪subsequent calls to "homing" function will bring it there.
*
* Note: homing settings are saved into RAM and will be lost when controller is␣
↪powered down. If you wish to save these settings to non-volatile memory you should␣
↪either pick "Save settings to flash" on the XILab main Settings page or call
↪"command_save_settings()" at the end of the script.
*
* To run the script, upload it to the XILab software
*/

var axes = [];
var number_of_axes = 0;
var last_serial = 0;
while (serial = get_next_serial(last_serial)) // get next serial number and repeat␣
↪for each axes.
{
    axes[number_of_axes] = new_axis(serial);
   log("Found axis " + number_of_axes + " with serial number " + serial);
   number_of_axes++;
   last_serial = serial;
}
```

```
for (var i = 0; i < number_of_axes; i++)
{
   set_zero(axes[i]);
}

function set_zero(axis)
{
    axis.command_stop(); // send STOP command (does immediate stop)

   var h = axis.get_home_settings(); // read homing settings from the controller
   h.HomeDelta = 0; // set "HomeDelta" parameter in "home_position" structure to 0
   h.uHomeDelta = 0; // set "uHomeDelta" parameter in "home_position" structure to 0
   var saved_fast = h.FastHome; // save "FastHome" parameter from "home_position"␣
↪structure to a variable
   var saved_ufast = h.uFastHome; // save "uFastHome" parameter from "home_position"␣
↪structure to a variable
   if (h.HomeFlags & HOME_MV_SEC_EN != 0) // if homing settings have two homing␣
↪phases turned on
   {
        h.FastHome = 100; // set "FastHome" parameter in "home_position" structure␣
↪(first movement speed) to 100 steps/s: this is required to avoid slip at the end of␣
↪the first phase
       h.uFastHome = 0; // set "uFastHome" parameter in "home_position" structure to 0
    }
    axis.set_home_settings(h); // write homing settings into the controller

    var old_pos = axis.get_status().CurPosition; // save whole step part of the␣
↪initial position into a variable
    var old_upos = axis.get_status().uCurPosition; // save microstep part of the␣
↪initial position into a variable
    axis.command_home(); // send HOME command (find home position)
    do { msleep(100); } while (axis.get_status().MvCmdSts == (MVCMD_HOME | MVCMD_
↪RUNNING)); // query controller state every 100 ms while movement state is "homing␣
↪command is being executed"
    if (axis.get_status().MvCmdSts != MVCMD_HOME)  // if current state is not "homing␣
↪completed successfully" (MVCMD_RUNNING unset, MVCMD_ERROR unset, last command MVCMD_
↪HOME) then homing was interrupted
    {
        h.FastHome = saved_fast; // set "FastHome" parameter in "home_position"␣
↪structure to a saved value
       h.uFastHome = saved_ufast; // set "uFastHome" parameter in "home_position"␣
↪structure to a saved value
       axis.set_home_settings(h); // write movement settings into the controller␣
↪(this restores the initial settings)
       throw "Script aborted: homing failed."; // throw an exception and terminate
    }

    var new_pos = axis.get_status().CurPosition; // read whole part of new position␣
↪into "new_pos" variable
    var new_upos = axis.get_status().uCurPosition; // read microstep part of new␣
↪position into "new_upos" variable
    h.HomeDelta = old_pos-new_pos; // set "HomeDelta" parameter in "home_position"␣
↪structure to this value
    h.uHomeDelta = old_upos-new_upos; // set "uHomeDelta" parameter in "home_position
↪" structure to this value
    h.FastHome = saved_fast; // set "FastHome" parameter in "home_position" structure␣
↪to a saved value
```

```
    h.uFastHome = saved_ufast; // set "uFastHome" parameter in "home_position"␣
↪structure to a saved value
    axis.set_home_settings(h); // write movement settings into the controller
    axis.command_move(old_pos, old_upos); // move to initial position
    do { msleep(100); } while (axis.get_status().MvCmdSts == (MVCMD_MOVE | MVCMD_
↪RUNNING)); // query controller state every 100 ms while movement state is "movement␣
↪command is being executed"
    if (axis.get_status().MvCmdSts != MVCMD_MOVE)  // if current state is not
↪"movements completed successfully" (MVCMD_RUNNING unset, MVCMD_ERROR unset, last␣
↪command MVCMD_MOVE) then movement was interrupted
    {
        throw "Script aborted: return to position failed."; // throw an exception and␣
↪terminate
    }
    axis.command_zero(); // send ZERO command (sets current position and encoder␣
↪value to zero)
}

log("Done."); // log success
```

6.5.4.10 Autotester script

```
/*
* Autotester script
*
* Description of the script:
*  The script tests the controller with a stage using a set of tests.
*
* Note: This is a rather difficult script to learn, since it uses a large number of␣
↪commands and structures.
*
* To run the script, upload it to the XILab software
*/
```

View the full code

6.5.4.11 Border crossing test

```
/*
* Border crossing test
*
* Description of the script:
*  The script checks the correct operation of the connected external limit switch
*
* How to connect wires?
*  You must connect the limit switch to the DSub-15 connector (pins 8 and 9 on the␣
↪controller connector).
*
* Note: This is a rather difficult script to learn, since it uses a large number of␣
↪commands and structures.
*
* To run the script, upload it to the XILab software
*/

const MVCMD_ERROR = 0x40;
const MVCMD_RUNNING = 0x80;
```

```
var axis = new_axis(get_next_serial(0));
var m = axis.get_extio_settings();
var s = axis.get_status();

var count_error = 0;
var count_good = 0;

function BorderOff()
{
  m.EXTIOSetupFlags = 0x01;
  m.EXTIOModeFlags = 0x10;

  axis.set_extio_settings(m);
}

function BorderOn()
{
  m.EXTIOSetupFlags = 0x01;
  m.EXTIOModeFlags = 0x00;

  axis.set_extio_settings(m);
}

function BorderCycle()
{
  BorderOn();
  msleep(50);

  BorderOff();
  msleep(100);
}

log("You have to connect the common",2);
log("input/output pin on the backplane connector to",2);
log("the 2nd limit switch pin on the stage connector", 2);
log("Also its recommended to load the profile for your stage", 2);

log(">>> Start testing", 3);

while (1)
{
  axis.command_left();
  msleep(200);
  BorderOn();
  msleep(200);
  s = axis.get_status();

  if (s.MvCmdSts & MVCMD_RUNNING)
  {
    count_error++;

    log(">>> ALARM ! Crossing through the limit switch !" ,1);
  }
  else
  {
    count_good++;

    if (!(count_good % 50))
```

```
      log(">>> " + count_good + " cycles were done correct, " + count_error + "␣
↪cycles were done incorrect", 3);
  }

  BorderOff();
}
```

6.5.4.12 Closed loop tuning test

```
/*
* Closed loop tuning test
*
* Description of the script:
*  The script checks the parameters of a closed loop
*
* Note: This is a rather difficult script to learn, since it uses a large number of␣
↪commands and structures.
*
* To run the script, upload it to the XILab software
*/

var global_axis;

global_axis = new_axis(get_next_serial(0));
global_axis.command_stop();

const DEBUG = 1;
const MICROSTEPS = 256;
const SKIP_ENCODER_COUNT = 7;
const STORE_COUNT_MICROSTEPS = 19;
const TRUST_INDEX = 15;     // Always TRUST_INDEX < STORE_COUNT_MICROSTEPS

/*
* Save and overwrite settings
*/
var SFBS = global_axis.get_feedback_settings(); // Save information about encoder␣
↪(IPS)
SFBS.FeedbackType = FEEDBACK_NONE;              // Overwrite feedback type, because␣
↪in profile feedback is encoder

var SENG = global_axis.get_engine_settings();   // Save information about engine␣
↪(nominal current, steps per revolution)

var SENT = global_axis.get_entype_settings();   // Save information about engine type

var SEDS = global_axis.get_edges_settings();    // Save information about edges

/*
* Clear FRAM for synchronization
* real full step with full step of firmware
*/
log("Clearing FRAM", 1);
global_axis.command_clear_fram();
msleep(4000);

/*
* Restore controller settings
```

```
*/
global_axis.set_feedback_settings(SFBS);
msleep(100);

global_axis.set_engine_settings(SENG);
msleep(100);

global_axis.set_entype_settings(SENT);
msleep(100);

global_axis.set_edges_settings(SEDS);
msleep(100);

/*
* Prepare and apply move settings
*/
var SMOV = global_axis.get_move_settings();
SMOV.Speed = 0;
SMOV.uSpeed = 16;
global_axis.set_move_settings(SMOV);
msleep(100);

/*
* Going to the full step and waiting 3 seconds for equilibration
*/
global_axis.command_move(0, 0);
msleep(3000);

/*
* Arrays for measurements and structure for GPOS
*/
var MicroStepsToRight = [];
var MicroStepsToLeft = [];
var EncToRight = [];
var EncToLeft = [];
var GPOS;

/*
* Start moving
*/
global_axis.command_right();

/*
* Skipping a some first counts for the stable experiment
*/
for (var i = 0; i < SKIP_ENCODER_COUNT; i++)
{
  GPOS = global_axis.get_position();
  var EncPos = GPOS.EncPosition;

  while (EncPos == GPOS.EncPosition)
  {
    msleep(40);
    GPOS = global_axis.get_position();
  }

  EncPos = GPOS.EncPosition;
}
```

```
/*
* Start measurements
*/
for (var i = 0; i < STORE_COUNT_MICROSTEPS; i++)
{
  GPOS = global_axis.get_position();
  var EncPos = GPOS.EncPosition;

  while (EncPos == GPOS.EncPosition)
  {
    msleep(40);
    GPOS = global_axis.get_position();
  }

  EncPos = GPOS.EncPosition;

  MicroStepsToRight[i] = GPOS.Position * MICROSTEPS + GPOS.uPosition;
  EncToRight[i] = GPOS.EncPosition;
}

/*
* Stop moving
*/
global_axis.command_stop();

/*
*  Start moving and measurements
*/
global_axis.command_left();

for (var i = 0; i < STORE_COUNT_MICROSTEPS; i++)
{
  GPOS = global_axis.get_position();
  var EncPos = GPOS.EncPosition;

  while (EncPos == GPOS.EncPosition)
  {
    msleep(40);
    GPOS = global_axis.get_position();
  }

  EncPos = GPOS.EncPosition;

  MicroStepsToLeft[STORE_COUNT_MICROSTEPS - 1 - i] = GPOS.Position * MICROSTEPS +␣
↪GPOS.uPosition;
  EncToLeft[STORE_COUNT_MICROSTEPS - 1 - i] = GPOS.EncPosition;
}

/*
* Stop moving
*/
global_axis.command_stop();

/*
* Check all values
*/
for (var i = 0; i < STORE_COUNT_MICROSTEPS; i++)
```

```
{
  var diffMicrosteps = MicroStepsToRight[i] - MicroStepsToLeft[i];
  var diffConts = EncToRight[i] - EncToLeft[i];

  if (DEBUG)
  {
    log(MicroStepsToRight[i] + " - " + MicroStepsToLeft[i] + " = " + diffMicrosteps +
↪" microstep(s)\t" +
      EncToRight[i] + " - " + EncToLeft[i] + " = " + diffConts + " count(s)", 3);
  }

  if (diffConts != 1)
  {
    log("Script error! Try again", 1);
  }
}

var RightB = (MicroStepsToRight[TRUST_INDEX] * EncToRight[0] - MicroStepsToRight[0] *␣
↪EncToRight[TRUST_INDEX]) / (EncToRight[0] - EncToRight[TRUST_INDEX]);
var LeftB = (MicroStepsToLeft[TRUST_INDEX] * EncToLeft[0] - MicroStepsToLeft[0] *␣
↪EncToLeft[TRUST_INDEX]) / (EncToLeft[0] - EncToLeft[TRUST_INDEX]);

log("Right counts show B = " + RightB, 2);
log("Left counts show B = " + LeftB, 2);
log("Aver B = " + ((RightB + LeftB) / 2), 3);
```

6.5.4.13 Discrete motion script

```
/*
* Discrete motion script
*
* Description of the script:
*  The script opens two axes by serial numbers. Moves along the X axis to a certain␣
↪coordinate. Then it begins to shift discretely along the Y axis, with a␣
↪programmable pause after each offset.
*  The script can be useful if you are using the system to scan an area and/or␣
↪capture frames
*
* Warning: enter the serial numbers of your axes!
*
* Note: This is a rather difficult script to learn, since it uses a large number of␣
↪commands and structures.
*
* To run the script, upload it to the XILab software
*/

// Enter the serial numbers of your axes.
serial_number_x = 14889;
serial_number_y = 14888;

var x = new_axis(serial_number_x);
var y = new_axis(serial_number_y);

// Installing the source data
var x_target_coordinate = 5000; // first border coordinate
var delay = 100; // delay in milliseconds
```

```
var y_first_border = 0; // first border coordinate
var y_second_border = 5000; // second border coordinate
var y_step = 100
var y_direct = 1

// Calibration and positioning of the axes to their original positions.
x.command_stop(); // send STOP command (does immediate stop)
x.command_zero(); // send ZERO command (sets current position and encoder value to␣
↪zero)
y.command_stop(); // send STOP command (does immediate stop)
y.command_zero(); // send ZERO command (sets current position and encoder value to␣
↪zero)

x.command_move(x_target_coordinate); // move to wards one border
x.command_wait_for_stop(10); // wait until controller stops moving
y.command_move(y_first_border); // move towards one border
y.command_wait_for_stop(10); // wait until controller stops moving

// Movement in discrete samples along one axis from the end to the end
// with a delay after each movement.
while (1) { // infinite loop

// Choosing the direction of travel
if (y.get_position() >= y_second_border)
{
  y_direct = -1
}
if (y.get_status().CurPosition <= y_first_border)
{
  y_direct = 1
}

// Movement in a given direction
y.command_movr(y_step*y_direct); // move towards another border
y.command_wait_for_stop(10); // wait until controller stops moving
msleep(delay);
}
```

6.5.4.14 Exponential position change in user units script

```
/*
* Exponential position change in user units script
*
* Description of the script:
*  The script performs discrete control of movement according to a certain law of␣
↪motion. The amplitude and the law of displacement are given. A correction speed is␣
↪used to maintain the positioning accuracy.
*
* Note: This is a rather difficult script to learn, since it uses a large number of␣
↪commands and structures.
*
* To run the script, upload it to the XILab software
*/

// Main characteristics
var time_discre = 10; // Discreteness of movement control(ms)
var end_err = 0.01; // The accuracy of reaching the final coordinate of the movement.
```

```
var full_move = 6; // Total movement in mm

// If you change the equation of motion, you will need to change the equation for the␣
↪correction velocity, since this change is not linear.
var K = 0.4;
var Glob_err = 0;

// Advanced setting.
var mm_per_step = 0.00125; // Distance in gr for 1 completed step.
var calb = new_calibration(mm_per_step, get_engine_settings().MicrostepMode); //␣
↪create calibration structure

// Setting the starting position.
command_stop(); // send STOP command (does immediate stop)
command_wait_for_stop(10); // wait until controller stops moving
command_zero(); // send ZERO command (sets current position and encoder value to zero)


log("Start:");
// Setting 0 speeds and accelerations.
zero_movesettings();

//
go_position(full_move, time_discre)


// Function for calculating the set speed and acceleration
function set_movesettings(time, corr_speed)
{
  // Speed, Accel, Decel setting.
  var m = get_move_settings_calb(calb); // read movement settings from the controller

  // The equation of speed is equal to the derivative of the equation of motion.
  m.Speed =  K*Math.exp(K*time/1000 )+ corr_speed;

  set_move_settings_calb(m, calb); // write movement settings into the controller

log("Speed = " +m.Speed);
log("corr_speed = " +corr_speed);
}

// Set the initial parameters of motion
function zero_movesettings()
{
  var m = get_move_settings_calb(calb); // read movement settings from the controller
  m.Speed = 0.1; // set movement speed

  set_move_settings_calb(m, calb); // write movement settings into the controller
}

// A function of calculating target coordinates from time
function current_target_coordinate(time)
{
  // The equation of motion. The time is set in milliseconds. The position at the␣
↪initial time is 0.
  return (Math.exp(K*time/1000) - 1);
}
```

```
// The calculation of the correction speed
function speed_corr(err_pos)
{
  Glob_err = Glob_err + err_pos*0.35;
  return  Glob_err;
}

// The main moving
function go_position(full_time, time_discre)
{
  Glob_err = 0;
  var end_position = full_move;

  // Setting the movement to the desired coordinate.
  command_move_calb(end_position, calb);

  // Pause before starting to move, to turn on the power button.
  msleep(300);
  var mas = 1;
  var basetime = new Date();
  var curr_time = new Date();
  var err_pos = 0;
  var pos = 0;
  var pos1 = 0;
  var i = time_discre;
  do {
    // If you do not need position feedback, you can instead speed_corr(err_pos)␣
↪write 0
    set_movesettings(i+1, speed_corr(err_pos));

    // Waiting for the end of a discrete time interval
    do {
      curr_time = new Date - basetime;
      msleep(1);
    }
    while ((curr_time) < i); //

    // Reading the actual and calculating the planned coordinate if used.
    pos =  get_position_calb(calb).Position;
    pos1 =  current_target_coordinate(i);

    // Calculation of the position error.
    err_pos =  (pos1 -  pos);

    log("time = " + i);
    log("err_pos = " + err_pos);

    i = i + time_discre
  }
  while (Math.abs(pos - end_position) > end_err); // Completion when the end of the␣
↪movement is reached with the specified accuracy.
}
```

6.5.4.15 For calb step script

```
/*
* For calb step script
*
* Description of the script:
*  In the script, the user units are configured, after which there is a departure to the leftmost position and then a certain number of shifts occur until the right position is reached. Each position is recorded in a .csv file.
*
* Note: This is a rather difficult script to learn, since it uses a large number of commands and structures.
*
* To run the script, upload it to the XILab software
*/

command_home(); // send HOME command (find home position)
command_wait_for_stop(100);
command_zero();
command_wait_for_stop(100);

// Setting the value to convert to user unit
var mm_per_step = 0.00125; // steps to distance translation coefficient
var calb = new_calibration(mm_per_step, get_engine_settings().MicrostepMode); // create calibration structure

// Setting boundaries and movement step
// Boundaries can be set manually or taken from limit constraints
var edge = get_edges_settings_calb(calb);
var first_border = edge.LeftBorder; //0;
var second_board = edge.RightBorder;//23;
var shift = 5;//step move;
var delay = 2000;// The delay of movement

var f = new_file("file.csv"); // Choose a file name and path
f.open(); // Open a file
f.resize(0);
f.seek( 0 ); // Seek to the beginning of the file

command_move_calb(first_border, calb); // Move to the starting position
command_wait_for_stop(delay); // Wait until controller stops moving
var i = 1;
var time = 0;
var step = (second_board - first_border)/shift;
f.write(0+ "," + get_status().CurPosition + "," + get_status().uCurPosition+ "," + "\n" ); // Get current position, potentiometer value and date and write them to file
do {
  time = i*delay;
  command_movr_calb(shift, calb); // Move to the next position
  command_wait_for_stop(delay); // Wait until controller stops moving
  f.write(time + "," + get_position_calb(calb).Position + "," + get_position_calb(calb).EncPosition+ "," + "\n" ); // Get current position, potentiometer value and date and write them to file
  i = i+1;
} while ( get_position_calb(calb).Position+shift < second_board )
f.close(); // Close the file
```

6.5.4.16 Step script

```
/*
* Step script
*
* Description of the script:
*  In the script, there is a departure to the leftmost position and then a certain␣
↪number of shifts occur until the right position is reached. Each position is␣
↪recorded in a .csv file.
*
* Note: The script is similar to the "for_calb_step" script, only in this script the␣
↪offset occurs at the step mode
*
* To run the script, upload it to the XILab software
*/

command_home(); // send HOME command (find home position)
command_wait_for_stop(100);
command_zero();
command_wait_for_stop(100);
var edge = get_edges_settings();
var first_border = edge.LeftBorder;
var second_board = edge.RightBorder;
var count = 10;
var f = new_file("E:/a.csv"); // Choose a file name and path
f.open(); // Open a file
f.seek( 0 ); // Seek to the beginning of the file

var delay = 2000;
command_move(first_border); // Move to the starting position
command_wait_for_stop(delay); // Wait until controller stops moving
var i = 1;
var time = 0;
var shift = 6;
var step = (second_board - first_border)/shift;
f.write(0+ ","  +  get_status().CurPosition + "," +  get_status().uCurPosition+ "," +
↪"\n" ); // Get current position, potentiometer value and date and write them to file
do {
  time = i*delay;
  command_movr(step, 0); // Move to the next position
  command_wait_for_stop(delay); // Wait until controller stops moving
  f.write(time + ","  +  get_status().CurPosition + "," +  get_status().uCurPosition+
↪"," + "\n" ); // Get current position, potentiometer value and date and write them␣
↪to file
  i = i+1;
} while (i < shift )
f.close(); // Close the file
```

6.5.4.17 Homing test with extio

```
/*
* Homing test with extio
*
* Description of the script:
*  The script starts calibration when a signal is received from a general purpose␣
↪digital input/output (extio)
*
* How to connect wires?
```

```
*  You must connect to the HDB-26 connector (pin 25 on the controller).
*
* Note: This is a rather difficult script to learn, since it uses a large number of␣
↪commands and structures.
*
* To run the script, upload it to the XILab software
*/

const MVCMD_ERROR = 0x40;
const MVCMD_RUNNING = 0x80;

var axis = new_axis(get_next_serial(0));
var m = axis.get_extio_settings();
var s = axis.get_status();

var count_error = 0;
var count_good = 0;

function BorderOff()
{
  m.EXTIOSetupFlags = 0x01;
  m.EXTIOModeFlags = 0x10;

  axis.set_extio_settings(m);
}

function BorderOn()
{
  m.EXTIOSetupFlags = 0x01;
  m.EXTIOModeFlags = 0x00;

  axis.set_extio_settings(m);
}

function BorderCycle()
{
  BorderOn();
  msleep(50);

  BorderOff();
  msleep(2500);
}

log(">>> Start testing", 3);

while (1)
{
  axis.command_home();
  msleep(1500);

  BorderCycle();
  BorderCycle();

  command_wait_for_stop(100);

  s = axis.get_status();

  if (s.MvCmdSts & MVCMD_ERROR)
```

```
  {
    count_error++;

    log(">>> Alarm! Homing broken", 1);
  }
  else
  {
    count_good++;

    if (!(count_good % 50))
      log(">>> " + count_good + " cycles were done correct, " + count_error + "␣
↪cycles were done incorrect", 3);
  }
}
```

6.5.4.18 Motion by sin function

```
/*
*  Motion by sin function
*
* Description of the script:
*  A script for moving with a change in speed according to the trigonometric law.
*  The script can be useful for precise positioning of a laser or motorized mirror
*
* Note: This is a rather difficult script to learn, since it uses a large number of␣
↪commands and structures.
*
* To run the script, upload it to the XILab software
*/

var delay = 100;

/* Definition of delta and initial phase*/
var df = 0.05;
var f = 0;

/* Definition of package */
var GETS = new Object();

// Initial installations
var move_set = get_move_settings();
var speed = 3000;
var amplitude = 1000;
var nomber = 100;
var time = 1000;
var pos = 0;
var Pi = 3.1415;
df = Pi/nomber;

command_zero();

var pos_read = new Object();
while (1)
{
    pos_read = get_position();

    // Movement to a point with an increase in the velocity amplitude
```

```
    for (i = 1; i <= nomber-1; i++)
    {
      f = df*i;

      move_set.Speed = speed * Math.sin(f);
      pos =  amplitude*i /nomber;

      set_move_settings(move_set);
      command_move(pos);
      while (Math.abs(pos_read.Position - pos)>move_set.Speed/10)
      {
        pos_read = get_position();
      }

    }

    // Movement to a point with a decrease in the velocity amplitude
    for (i = nomber-1; i >= 1; i--)
    {
      f = df*i;

      move_set.Speed = speed * Math.sin(f);
      pos =  amplitude*i /nomber;

      set_move_settings(move_set);
      command_move(pos);
      while (Math.abs(pos_read.Position - pos)>move_set.Speed/10)
      {
        pos_read = get_position();
      }

    }

  msleep(1000);

}
```

6.5.4.19 Move EXTIO calb script

```
/*
* Move EXTIO calb script
*
* Description of the script:
*  The script moves to one of the 2 specified points, depending on the state of the␣
↪EXTIO input. The movement is carried out in user units.
*
* Note: This is a rather difficult script to learn, since it uses a large number of␣
↪commands and structures.
*
* To run the script, upload it to the XILab software
*/

// Main characteristics
var time_discre = 10; // Discreteness of movement control(ms)
var nomspeed = 5;
var end_err = 0.015;
```

```
var low_position = 0; // Move to position for low EXTIO
var high_position = 180; // Move to position for high EXTIO

// Advanced setting.
var gr_per_step = 0.015; // Distance in gr for 1 completed step.
var calb = new_calibration(gr_per_step, get_engine_settings().MicrostepMode); //␣
↪create calibration structure

// Setting the starting position.
command_stop(); // send STOP command (does immediate stop)
command_wait_for_stop(10); // wait until controller stops moving
command_home();
command_zero(); // send ZERO command (sets current position and encoder value to zero)


log("Start:");
// Setting 0 speeds and accelerations.
movesettings();

extiosettings();
//
go_position(time_discre)

function extiosettings()
{
  var extsettings = get_extio_settings();
  extsettings.EXTIOSetupFlags = 0x00;
  extsettings.EXTIOModeFlags = 0x00;
  set_extio_settings(extsettings);
}

// Set the initial parameters of motion
function movesettings()
{
  var m = get_move_settings_calb(calb); // read movement settings from the controller
  m.Speed = nomspeed; // set movement speed

  set_move_settings_calb(m, calb); // write movement settings into the controller
}


// The main moving
function go_position(time_discre)
{
  var oldstate = 0;
  var maskstate = 32;

  // Setting the movement to the desired coordinate.
  command_move_calb(low_position, calb);

  // Pause before starting to move, to turn on the power button.
  msleep(300);

  while(1) {

  //
  var status = get_status_calb(calb);
  if ((status.GPIOFlags & maskstate) != oldstate)
```

```
  {
    log(status.GPIOFlags);
    if (oldstate)
      command_move_calb(high_position, calb);
    else
      command_move_calb(low_position, calb);

    oldstate = status.GPIOFlags & maskstate;
  }
  // Waiting for the end of a discrete time interval
  msleep(time_discre);
  }
}
```

6.5.4.20 Probabilistic tests

```
/*
* Probabilistic tests
*
* Description of the script:
*  The script runs a set of repeatable tests a certain number of times and is␣
↪expected to fail.
*
* Note: This is a rather difficult script to learn, since it uses a large number of␣
↪commands and structures.
*
* To run the script, upload it to the XILab software
*/
```

View the full code

6.5.4.21 Several shifts with calibration script

```
/*
* Several shifts with calibration script
*
* Description of the script:
*  This program makes shifts given number of times to the specified distance, and␣
↪stands the appointed time after every shift. First it goes left, then it returns␣
↪back to the origin and repeats all movements to right.
*
* Note: This is a rather difficult script to learn, since it uses a large number of␣
↪commands and structures.
*
* To run the script, upload it to the XILab software
*/

const LEFT = -1;
const RIGHT = 1;

var axes = [];
var number_of_axes = 0;
var last_serial = 0;
while (serial = get_next_serial(last_serial)) // get next serial number and repeat␣
↪for each axes.
{
    axes[number_of_axes] = new_axis(serial);
```

```
    log("Found axis " + number_of_axes + " with serial number " + serial);
    number_of_axes++;
    last_serial = serial;
}

// Start the main function for all avaliable axes
for (var i = 0; i < number_of_axes; i++)
{
    main(axes[i]);
}


function main(axis)
{
    axis.command_move(0,0); // Go back to the origin.
    msleep(500);

/* Creating and filling the calibration structure for specifying distances in um */
    var calibration = new Object;
    calibration.A = 5; // 1 step correspond to 5 um for 8MT50-100BS1
    calibration.MicrostepMode = axis.get_engine_settings().MicrostepMode; // Get␣
↪MicrostepMode from controller settings
/*******************************************************/

/* Main cycle */

    var N = 10; // Number of shifts
    var stand_time  = 3000; // Stand time in ms
    var shift = 10; // Distance of shift in um

    MakeShifts(axis, LEFT,   shift, N, stand_time, calibration); // Make 10 shifts to␣
↪the left
    MakeShifts(axis, RIGHT, shift, N, stand_time, calibration); // Make 10 shifts to␣
↪the right
    MakeShifts(axis, RIGHT, shift, N, stand_time, calibration); // Make 10 shifts to␣
↪the right again
    MakeShifts(axis, LEFT,   shift, N, stand_time, calibration); // Make 10 shifts to␣
↪the left
}

function MakeShifts(axis, Direction, ShiftDistance, ShiftsQuantity, StandTime,␣
↪Calibration)
{
/**
    This function makes shifts which number is specified by ShiftQuantity and length␣
↪is specified by ShiftDistance. After every shift it stands StandTime miliseconds.␣
↪Calibration parameter is a structure for convertation between steps and micrometers.
*/
    for (var i = 0; i < ShiftsQuantity; i++)
    {
        axis.command_movr_calb(Direction*ShiftDistance, Calibration);
        axis.command_wait_for_stop(100);
        msleep(StandTime);
    }
}
```

6.5.4.22 Steps loss test

```
/*
* Steps loss test
*
* Description of the script:
*  The script was written to check for skipping steps
*  It can be useful for diagnosing problematic stages
*
* Note: This is a rather difficult script to learn, since it uses a large number of␣
↪commands and structures.
*
* To run the script, upload it to the XILab software
*/

function abs(x)
{
    return ((x > 0) ? x : -x);
}

/*
* Set home settings
*/
var SHOM = get_home_settings()
SHOM.FastHome = 500;
SHOM.uFastHome = 0;
SHOM.SlowHome = 20;
SHOM.uSlowHome = 0;
SHOM.HomeDelta = 300;
SHOM.uHomeDelta = 0;
SHOM.HomeFlags = HOME_STOP_FIRST_LIM;
set_home_settings(SHOM);

/*
* Check for encoder
*/
var encoder = 1
command_zero()
var first = get_status().EncPosition
command_movr(100)
command_wait_for_stop(300)
var second = get_status().EncPosition
if(abs(second - first) < 2)
{
  encoder = 0
  log("It seems like here are no encoder", 2)
}

command_home();
command_wait_for_stop(300);
command_zero();
msleep(200);
// Store old move settings
var MOV = get_move_settings();

// Set fast speed and accel\decel
var MOV2 =get_move_settings();
MOV2.Speed = MOV.Speed * 2;
MOV2.Accel = MOV.Accel * 2;
```

```
MOV2.Decel = MOV.Decel * 2;

for(var i=0; i < 1; i++) // Set default settings first
{
  set_move_settings(MOV2);

  // Move long...
  command_move(20000);
  command_wait_for_stop(300);

  // Set prev settings back
  set_move_settings(MOV);

  // Move back
  command_move(0);
  command_wait_for_stop(300);
}

if (encoder > 0)
{
  var lost = get_status().EncPosition
  if (abs(lost) > 1)
  {
    log("Lost " + lost + " pulses", 1)
  }
  else
  {
    log("All is OK");
  }
}
else
{
  log("Do final homing...")
  command_home()
  command_wait_for_stop(300)
  var  lost = get_status().CurPosition
  if (abs(lost) > 2)
  {
    log("Lost " + lost + " steps", 1)
  }
  else
  {
    log("All is OK");
  }
}
```

6.5.4.23 Sync test script

```
/*
* Sync test script
*
* Description of the script:
*  The script is written to demonstrate the work of synchronization, and also checks␣
↪its operability
*
* For test you must short special pins on the controller. Pin 5 (sync in) and pin 6␣
↪(sync out). See https://doc.xisupport.com/en/8smc5-usb/8SMCn-USB/Technical_
↪specification/Appearance_and_connectors/One_axis_system.html
```

```
*
* Note: This is a rather difficult script to learn, since it uses a large number of␣
↪commands and structures.
*
* To run the script, upload it to the XILab software
*/

const dX = 4.5;
const dot_num = 5;
const micro2mili = 1000;

var ASIA = [];

for (var i = 0; i < dot_num; i++)
  ASIA[i] = new Object();

var calb = new_calibration(1, MICROSTEP_MODE_FRAC_256);

function abs(x)
{
  return (x > 0) ? x : -x;
}

function set_default()
{
  // SSNI settings
  var SSNI = get_sync_in_settings_calb(calb);
  SSNI.SyncInFlags = SYNCIN_ENABLED | SYNCIN_GOTOPOSITION;
  set_sync_in_settings_calb(SSNI, calb);

  // SSNO settings
  var SSNO = get_sync_out_settings(calb);
  SSNO.SyncOutFlags = SYNCOUT_ENABLED;
  SSNO.Accuracy = 0.5;
  set_sync_out_settings(SSNO, calb);

  // SMOV settings
  var SMOV = get_move_settings_calb(calb);
  SMOV.Speed = 1000;
  SMOV.Accel = 500;
  SMOV.Decel = 1000;
  SMOV.AntiplaySpeed = 50;
  set_move_settings_calb(SMOV, calb);

  // SENG settings
  var SENG = get_engine_settings();
  SENG.NomSpeed = 5000;
  SENG.EngineFlags = ENGINE_ACCEL_ON | ENGINE_LIMIT_VOLT | ENGINE_LIMIT_CURR;
  SENG.Antiplay = 50;
  SENG.MicrostepMode = MICROSTEP_MODE_FRAC_256;
  SENG.StepsPerRev = 200;
  set_engine_settings(SENG);
}

function send_all_asia()
{
  for (var i = 0; i < dot_num; i++)
    command_add_sync_in_action_calb(ASIA[i], calb);
```

```
}

function check_all_asia()
{
  var GETS;

  for (var i = 0; i < dot_num; i++)
  {
    log("> Checking movement ASIA[" + i + "]", 3);
    msleep(ASIA[i].Time / micro2mili);
    GETS = get_status_calb(calb);
    if (abs(GETS.CurPosition - ASIA[i].Position) < dX)
      log(">>> OK! GETS.CurSpeed = " + GETS.CurSpeed, 3);
    else
      log(">>> Error! GETS.CurPosition = " + GETS.CurPosition + ", ASIA[" + i + "].
↪Position = " + ASIA[i].Position, 1);
    }
}

function test1()
{
  ASIA[0].Position = 22.5;
  ASIA[0].Time = 300000;

  ASIA[1].Position = 45.0;
  ASIA[1].Time = 300000;

  ASIA[2].Position = 32.5;
  ASIA[2].Time =300000;

  ASIA[3].Position = 10;
  ASIA[3].Time = 300000;

  ASIA[4].Position = -11.5;
  ASIA[4].Time = 300000;
}

function test2()
{
  ASIA[0].Position = -22.5;
  ASIA[0].Time = 300000;

  ASIA[1].Position = -45.0;
  ASIA[1].Time = 300000;

  ASIA[2].Position = -32.5;
  ASIA[2].Time =300000;

  ASIA[3].Position = -10;
  ASIA[3].Time = 300000;

  ASIA[4].Position = 11.5;
  ASIA[4].Time = 300000;
}

function test3()
{
  ASIA[0].Position = -6;
```

```
  ASIA[0].Time = 300000;

  ASIA[1].Position = 6;
  ASIA[1].Time = 300000;

  ASIA[2].Position = -6;
  ASIA[2].Time =300000;

  ASIA[3].Position = 6;
  ASIA[3].Time = 300000;

  ASIA[4].Position = -6;
  ASIA[4].Time = 300000;
}

command_zero();
set_default();
log(">>> Function test1() started", 3)
test1();
send_all_asia();
check_all_asia();
msleep(500);

command_zero();
set_default();
log(">>> Function test2() started", 3)
test2();
send_all_asia();
check_all_asia();
msleep(500);

command_zero();
set_default();
log(">>> Function test3() started", 3)
test3();
send_all_asia();
check_all_asia();
msleep(500);
```

6.5.4.24 Sync bug test script

```
/*
* Sync bug test script
*
* Description of the script:
*  This script does synchronization test to find sync bug in firmware 3.9.9, 3.8.7, 3.
↪9.10.
*  This test only for hardware from ver. 2.2.0 to ver. 2.2.4.
*  To begin the test, the first what you need to do is connect synchronization pin 15␣
↪to pin 16 on CN1 jack.
*  The scripts automatically set all necessary settings for test, and automatically␣
↪beginning the test.
*  As a result you will see a log message with current position of motor and text OK!␣
↪(green text) or ERROR! (red text).
*
* Note: This is a rather difficult script to learn, since it uses a large number of␣
↪commands and structures.
```

```
*
* To run the script, upload it to the XILab software
*/

var axes = [];
var number_of_axes = 0;
var last_serial = 0;

while (serial = get_next_serial(last_serial)) // get next serial number and repeat␣
↪for each axes.
{
  axes[number_of_axes] = new_axis(serial);
  number_of_axes++;
  last_serial = serial;
  log("Found axis " + number_of_axes + " with serial number " + serial);
}

// SSNI settings
var SSNI = get_sync_in_settings;
SSNI.SyncInFlags = SYNCIN_ENABLED | SYNCIN_GOTOPOSITION; //set flags for␣
↪synchronization in for enabled and absolute position
SSNI.ClutterTime = 4;// set sync in Clutter Time
SSNI.Position = 0;// set sync in absolute position
SSNI.Speed = 500;// set sync in speed
set_sync_in_settings(SSNI);// set synchronization in settings

// GSNO settings
var GSNO = get_sync_out_settings;
GSNO.SyncOutFlags = SYNCOUT_ENABLED | SYNCOUT_ONPERIOD;  // set flags for␣
↪synchronization out for enabled out and period
GSNO.SyncOutPeriod = 200;// set sync out period
GSNO.SyncOutPulseSteps = 2000;// set sync out Pusle width
set_sync_out_settings(GSNO);  // set synchronization out settings

for (i = 0; i < 10; i++)  // testing cycle
{
  GETS = get_status();// get information about device
  command_movr(201, 0);  // shifting position for 201 steps
  command_wait_for_stop(100); // Wait until the movement is complete

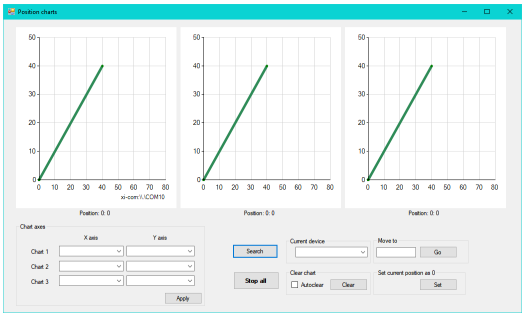
  if (GETS.CurPosition != 0)// get device current position and compare it with 0
  {
    log(">>> Error! GETS.CurPosition = " + GETS.CurPosition, 1);// log ERROR, bug is␣
↪exist
  }
  else
  {
    log(">>> OK! GETS.CurPosition = " + GETS.CurPosition, 3); // log OK, bug does not␣
↪exist
  }
}
```

6.6 Community examples

- *Example of six-axis XILab*
- *Examples for working with an attenuator for Python and LabView*
- *Program for sending commands to XIMC controller from AVR controller*
- *The multi-axis interface in Python*

Important: Below there are examples found on the open source Internet The developer is responsible for executing the examples

6.6.1 Example of six-axis XILab



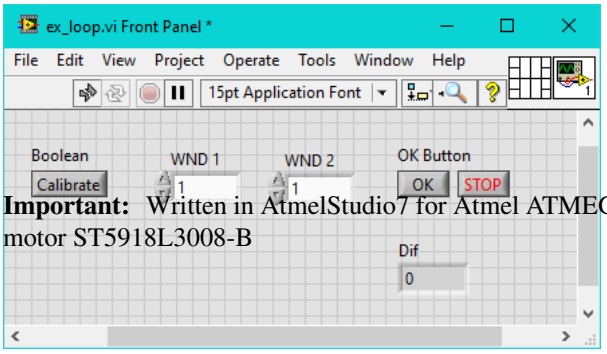
The program allows to operate with up to 6 devices in real time on 3 charts **The precompiled examples were build with Visual Studio 2013**.

Example and guide posted on GitHub: https://github.com/sushchev/testwfa.

6.6.2 Examples for working with an attenuator for Python and LabView

In this project several simple examples of using LibXIMC protocol with 8SMC4/5-USB controller were implemented. A rotating optical attenuator was used as a controlled device. *Code samples were written on Python 3.4 and also in LabVIEW programming environment*.

Example and guide posted on GitHub: https://github.com/Negrebetskiy/Attenuator.



6.6.3 Program for sending commands to XIMC controller from AVR controller

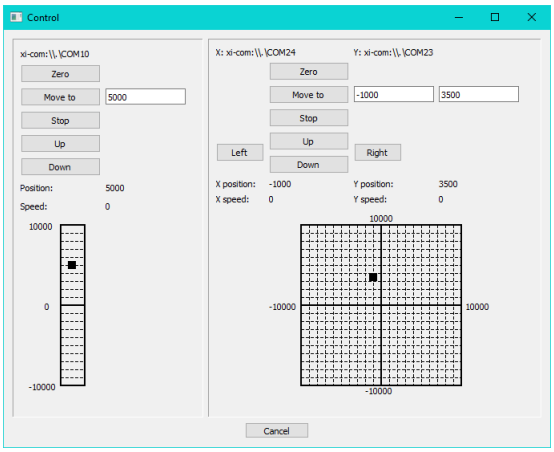
Important: Written in AtmelStudio7 for Atmel ATMEGA2560 16AU 1432 and used with XIMC controller and step motor ST5918L3008-B

What does this program:

- Gets current position of the step motor.
- Moves it to the left for 3 seconds and stops the step motor.
- Moves it to the init position

Example and guide posted on GitHub: https://github.com/ntarabrina/XIMC_command.

6.6.4 The multi-axis interface in Python



An example of a program with GUI, designed to work with multiple servo controllers (up to six inclusive, it is easy to increase this number if necessary) by means of XIMC protocol. GUI was created through PyQt5 package.

Example and guide posted on GitHub: https://github.com/Negrebetskiy/MultiAxleGUI.

CHAPTER

SEVEN

RELATED PRODUCTS

7.1 Control via Ethernet

7.1.1 Ethernet adapters Overview

Note: The **8SMC4-USB-Eth** adapter has been renamed **8Eth1**[1]. All available instructions in our manual are relevant and applicable to both devices.

7.1.1.1 General information



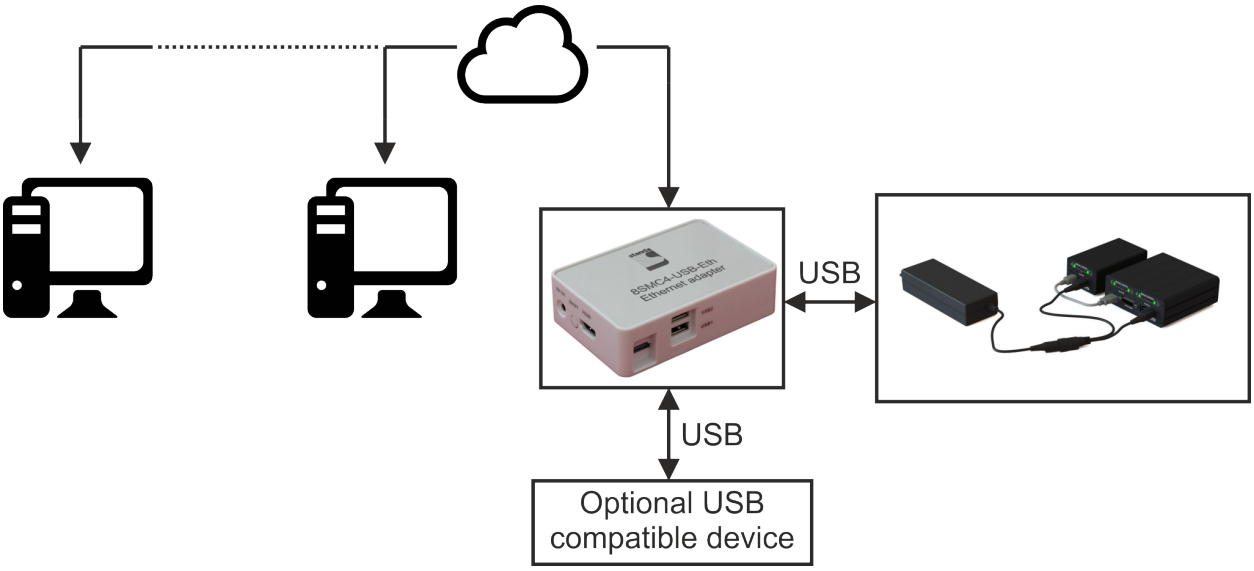
Fig. 7.1: Outward appearance of 8Eth1 adapter

8Eth1[1] is a universal device based on Cubieboard2 single-board computer with OS Linux inside, which enables the end user to interact with motor controllers remotely via various Ethernet-oriented interfaces. This device is also intended to be a universal provider for different services related to controllers' popular usage scenarios (e.g. motion control). It means that if there is a standalone program which could be effectively used in conjunction with our controllers, chances are high that this adapter already has it preconfigured and all you need is to plug some controllers and connect Ethernet cable to it. Out-of-the-box **8Eth1** supports online web-camera streaming, LAN auto-discovery system and two interfaces to configure and control motor controllers:

1. *TANGO* control system interface.
2. XiLab- and libximc-oriented *XIMC* interface.

Moreover **8Eth1** is equipped with embedded web-based *administration interface*, which enables the end user to conveniently control the device and monitor its state.

[1] The vendor code was changed in 2020, earlier this product had the vendor code 8SMC4-USB-Eth.



The appearance of the system from different perspectives (all necessary connectors in current version are marked with bold):



Fig. 7.2: Front view. Left to right: **power connector**, **on/off switch**, HDMI connector



Fig. 7.3: View from the right. Left to right: micro-SD card connector, **two USB type A female connectors**



Fig. 7.4: Back view. Left to right: **Ethernet connector**, mini-USB type B, switch to transfer into FEL mode, earphone and line in connectors



Fig. 7.5: View from the left. IR receiver

Using a USB hub, you can connect 36 controllers to this ethernet adapter. **IMPORTANT**: For a USB connection, it is better to use a tree structure. The fewer USB hubs you use, the better.

Important: The work with the multi-axis controller of 8SMC5-ETHERNET/RS232-B19 via Ethernet does not differ from the work with an 8Eth1 adapter

---

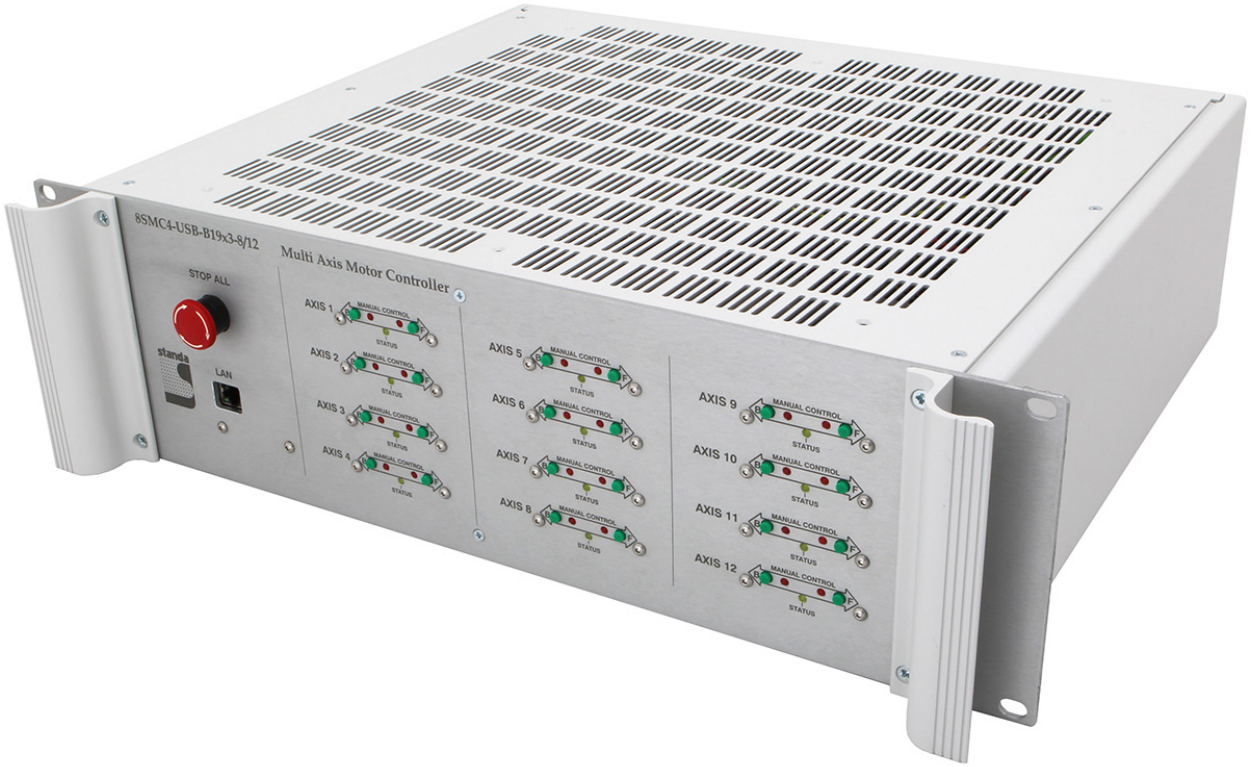


Fig. 7.6: Outward appearance multi-Axis Motion Controller of 8SMC5-ETHERNET/RS232-B19 (front view)



Fig. 7.7: Outward appearance multi-Axis Motion Controller of 8SMC5-ETHERNET/RS232-B19 (rear view)

Industrial motion controller/driver 8SMC4-ETHERNET/RS232-B19 is designed especially to control up to 12 mechatronic systems based on standard DC, Servo or Stepper technologies. Controller/driver housing can be mounted in standard 19 inch industrial cabinets. Several boxes can be connected in a single network via Ethernet port. System can be automatically set on a peer-to-peer mode without any special infrastructure or master servers to operate. Also each axis could be controlled via separate RS232 port.

8SMC4-ETHERNET/RS232-B19 contains all necessary subsystems, including: control, power units and etc., which support the simultaneous work of all axes. Controller/driver can be connected to a number of PC-based workstations via Ethernet. All users have access to the range setting of the available axes and assignment of access privileges. From the programmer's point of view, working with all axes of the system is very similar to working with a simple 8SMC4-USB box. The number of the axes available for a user and processed by the system can reach 50000. Front panel contains LED power and status indicators, manual buttons, system interlock button and Ethernet port. On the back panel there are motor connectors and separate RS232 ports for each axis, synchronization I/O connectors, a power connector and on/off button.The system is cooled through grille slots in upside and underside walls of the box.

7.1.1.2 Main requirements

7.1.1.2.1 Network configuration

Connecting the controller via a local network:

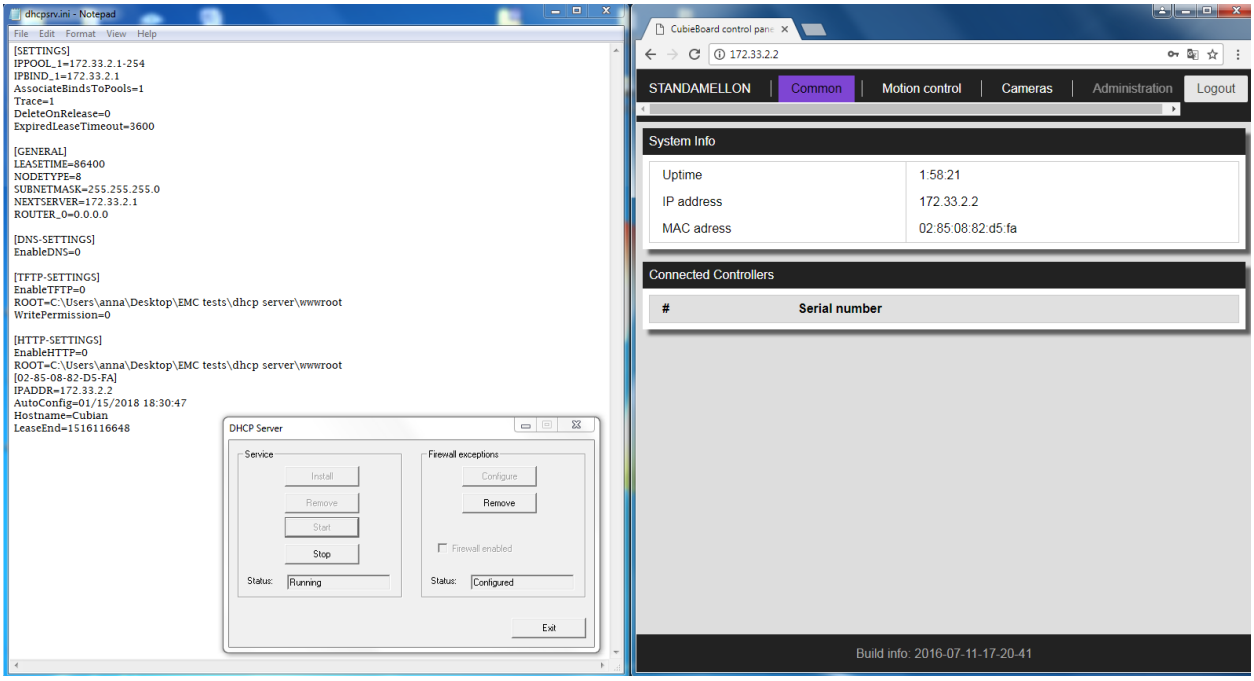
- You must have a DHCP server that supports an automatic distribution of ip addresses. The simple DHCP server that satisfies all requirements can be downloaded here.
- Server and your computer must support IPv4 protocol.
- Port 49150 should not be blocked. The cause of blocking can often be the presence of antivirus software, or program which monitors and filters network traffic (firewalls).

Warning: The Ethernet controller uses the multicast and UDP requests so make sure that such requests are enabled in your local network. The most common cause for the problem with the Ethernet controller detection is

the blocking of such requests by Windows Firewall or managed network equipment.

The Ethernet controller can also be used with the direct connection to the PC with working DHCP-server. The example of configuring the local connection for Windows OS:

- Download the simple DHCP server.
- Use the default config file for that server (lines 19 to 31 may be different) or configure the DHCP by yourself.
- Use *revealer* to find the given controller's ip and open it in the web browser.



Note: If necessary, you can use a static IP address. You can download the manual and image with a static IP address here.

7.1.1.2.2 Other

- 220V socket should be available.
- Ethernet cable, USB cable.

7.1.2 Administration

Note: Administration interface and *TANGO* support might not be present in your version of the device. You can update the device firmware using our *instruction*.

Note: The **8SMC4-USB-Eth** adapter has been renamed **8Eth1**[1]. All available instructions in our manual are relevant

[1] The vendor code was changed in 2020, earlier this product had the vendor code 8SMC4-USB-Eth.

and applicable to both devices.

7.1.2.1 Automatic device detection

As a convenience we provide a small dedicated utility called "Revealer" in order to help you instantly identify all the **Standa 8Eth1** adapters connected to your local network. You can download it from our software page.

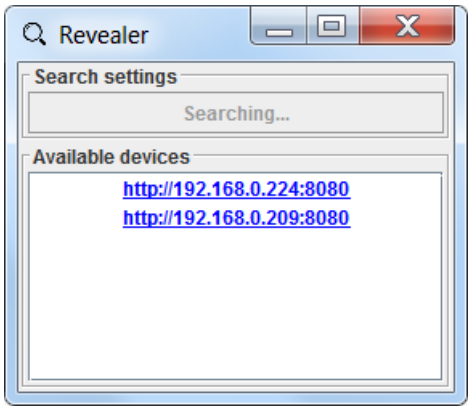


Fig. 7.8: "Revealer" utility interface

The Graphical User Interface of the "Revealer" is a simple one. In order to start search click *Search* button on *Search settings* panel - the scan takes approximately 3 seconds. After that all **Standa 8Eth1** devices found in your local network will be listed on *Available devices* panel as clickable links. When clicked, the link opens your defaul system browser and redirects it to Administartion interface web page.

The utility requires Java Runtime Environment (or simply JRE) version 6 or greater to work. There are high chances that you already have it installed on your PC as it's the requirement for a great number of popular software packages and so you just need to double-click the "revealer-j_0.1.0.jar" file to launch the utility.

Otherwise it means that you don't have JRE installed and have two options:

1. Download our ready-made packages for your operating system containing the "Revealer" utility and all the stuff necessary to make it work. In this case you just need to launch "Revealer" executable.
2. Install JRE. A good candidate is Oracle JRE which you can install following the official instructions.

Warning: "Revealer" uses UDP broadcasts to reach all **Standa 8Eth1** in your LAN so it might be unusable in the environments where UDP broadcasts are forbidden or unwanted.

7.1.2.2 Overview

Standa 8Eth1 and **Multi-Axis Motion Controller 8SMC4-ETHERNET/RS232-B19** devices is equipped with web-based Administration interface which enables the end user to control devices services and monitor system state.

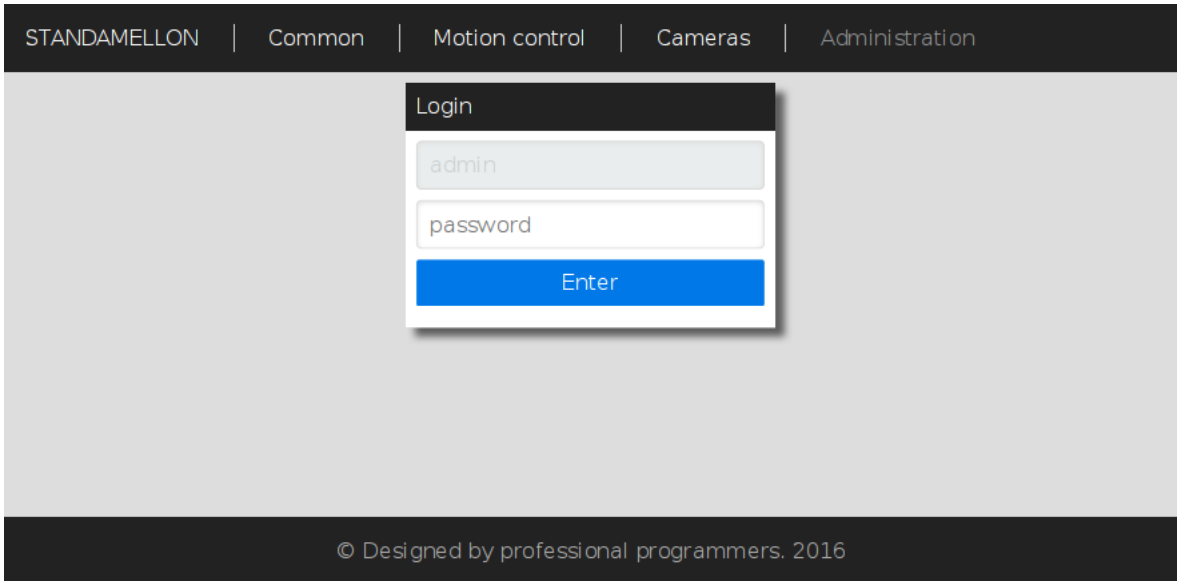


Fig. 7.9: Administration interface login page

In order to get access to the Administration panel, navigate your browser to *http:///[address]* URL (where *[address]* should be replaced with IP address of the device in your local network and can be obtained with the help of "Revealer" utility). If you are doing this for the first time (or you've disabled cookies/password storage in your browser) you'll need to authenticate yourself using "admin" as login and password.

Note: It's highly recommended to enable javascript in your browser while using Administration interface to get the best user experience.

Note: Web-interface is compatable with MS IE 9, MS IE 10, MS IE 11, MS Spartan, Firefox, Chrome browsers and might not work as expected in others.

Administration interface is split into three functional sections.

7.1.2.2.1 “Common” section

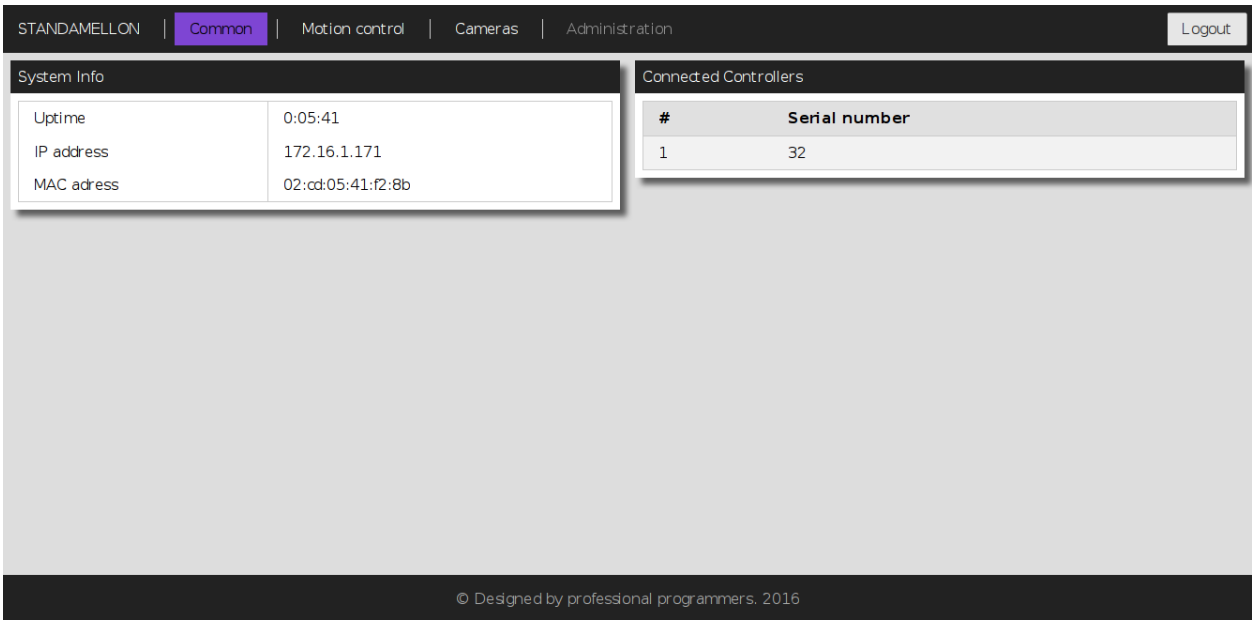


Fig. 7.10: Common section page

This section contains generic system information and decimal serial numbers of all controllers connected to the device.

7.1.2.2.2 “Motion control” section

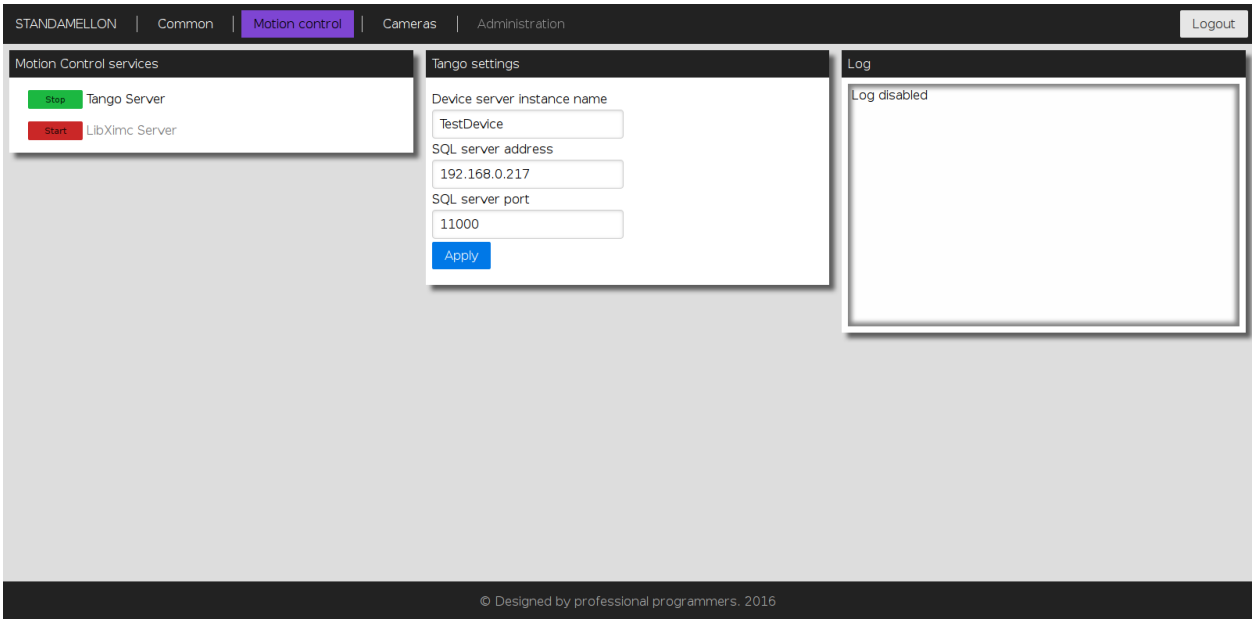


Fig. 7.11: Motion control section page

This section is dedicated to motion control services supported by the device. “Motion Control services” panel contains a list of all currently available motion control services. Clicking on services’ title opens an appropriate settings panel (or none if service has no alterable settings). Clicking “Apply” button in settings panel will persist settings in database and restart corresponding service.

Note: Motion control services are mutually exclusive - only one can be active at any given moment. Starting one of the services will automatically stop previous active if any.

7.1.2.2.3 “Cameras” section

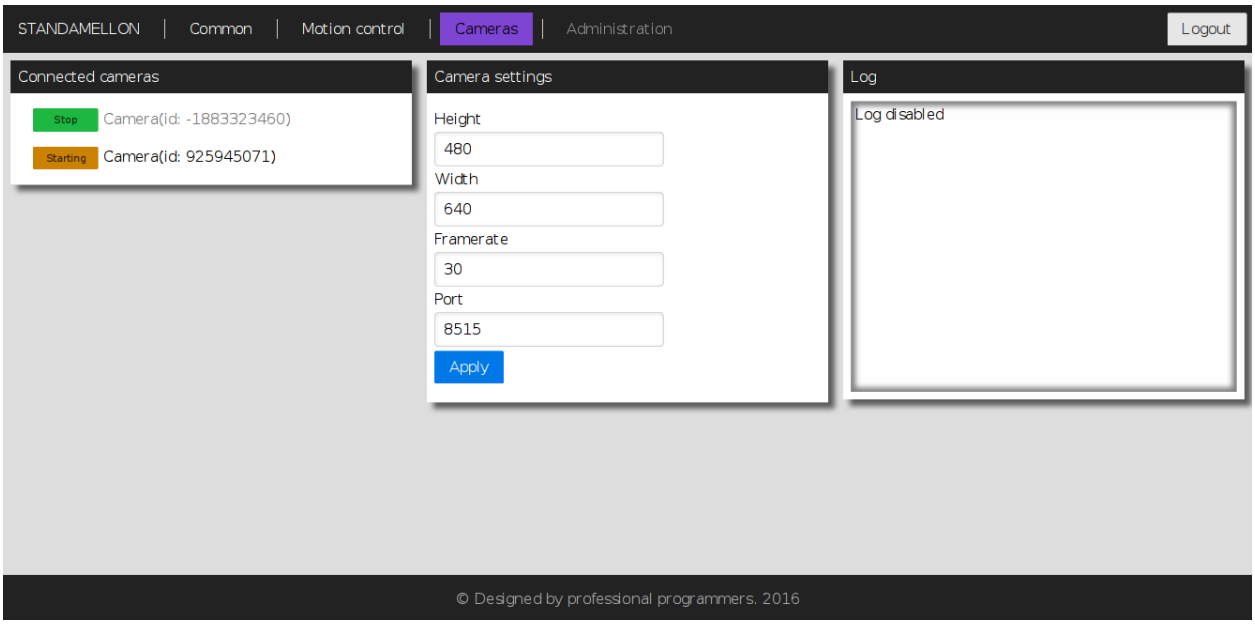


Fig. 7.12: Cameras section page

In this section there is a list of all connected cameras. Each camera is associated with streaming server instance, which exposes settings, status indicators and controls.

7.1.2.3 Service control

The Administration panel enables you to control all available device services. There is a common management metaphor for all services independent of its’ actual functionality and aim. Each service has an associated Indicator button on the left of its title which shows current services’s status and enables changing it:

- Red color with “Stopped” label indicates that service is completely stopped; pressing the button will run service startup.
- Orange color with “Starting” label indicates that service is in interruptible start sequence; pressing the button will revert service to stopped state.
- Green color with “Stop” label shows that the service is up and running; pressing the button will trigger service stop sequence.
- Orange color with “Stopping” label indicates uninterruptible stop sequence.

Note: Many services have settings which can be altered. After changing any of such settings the corresponding process will be automatically restarted shortly - the Indicator button will show it dynamically.

7.1.2.4 Firmware upgrade

Warning: During the update MicroSD content and 8Eth1 device flash memory will be wiped out. Make sure that you've made necessary backups.

1. Download an appropriate firmware version from software page.
2. Get a 4GB (or larger) MicroSD card.
3. Download and upack an autoinstaller image archive. On Windows family operating system you might have no apropriate archiver — in this case you may use 7-Zip (official site).
4. Put the image onto the MicroSD card:
   - **Unix family OS**:
     (a) You can do it via dd utility (e.g.

     ```
     dd if=autoinstall_image_2015-12-17.bin of=/dev/sdX bs=4M; sync
     ```

     but remember to substitute the device names with yours).
   - **Windows family OS**:
     (a) Download Win32 Disk Imager (official site) and start it.
     (b) Specify autoinstaller image path in "Image File" field.
     (c) Select your MicroSD drive in "Device" list.
     (d) Press "Write" button and wait till progressbar completes.
5. Insert the microSD card with the image into your **8Eth1** and turn it on (or reboot).
6. Wait approximately 10 minutes to let the firmware be installed. You may check the progress of installation by looking at LED indicator inside the case of the device through USB plugs:
   (a) LED is stable green or emits periodic single green pulses - autoinstaller is initializing.
   (b) LED periodically emits green double pulses - installation is in progress.
   (c) LED is stable green - installation is complete.
7. Power off the device (just plug the power cable off) and extract MicroSD card.
8. Power on the device and wait another 2-7 minutes to let the device initialize itself.

Warning: On some rare occasions the device might not be visible through the *revealer* (and have its services and administration interface unavailable either) even well after estimated initialization period. In this case try to reboot the device. If it doesn't help try to redo steps from 5 to 8.

Now you should have your **8Eth1** device updated and fully operational.

7.1.2.5 Troubleshooting

Warning: If your device won't work properly, leds don't blink as described above during flashing and your device manufactured **after 2017 October**, please, contact support.

7.1.2.5.1 Uninitialized Cubieboard2 bootloader

In some rare cases the steps described in Firmware upgrade instruction might not succeed. One of the possible reasons is that due to some technical failure the Cubieboard2 singleboard computer bootloader (on which **8Eth1** is based on) was not properly initialized. However bootloader initialization isn't very complicated:

1. Download PhoenixSuit (official site) and unpack it;
2. Download Lubuntu server image with bootloader supporting SoC A20 rev. for PhoenixSuit (official site, "Firmwares" -> "Cubieboard2" -> "Lubuntu Server") and unpack it.
3. Start PhoenixSuit. In case of update request, confirm it and wait till it's done.
4. Switch to "Firmware" tab and specify unpacked Lubuntu server image path.
5. Power off the device. Press and hold FEL button (it's near LAN and USB-OTG connectors). While still holding the button, connect USB-OTG to your PC and then release the button.
6. You might need to explicitly install driver for the device on Windows family OS.
   (a) Navigate to the Device Manager.
   (b) Locate "Unknown Device" and right click on it.
   (c) In the context menu press "Update driver software...".
7. Specify "**PhoenixSuit1.0.7\PhoenixSuit\Drivers\AW_Driver**" as driver search directory.
8. After that a popup in PhoenixSuite will appear. Press "YES (Full install)" and wait for the completion.
9. Continue to autoinstall instruction.

CHAPTER

EIGHT

FAQ

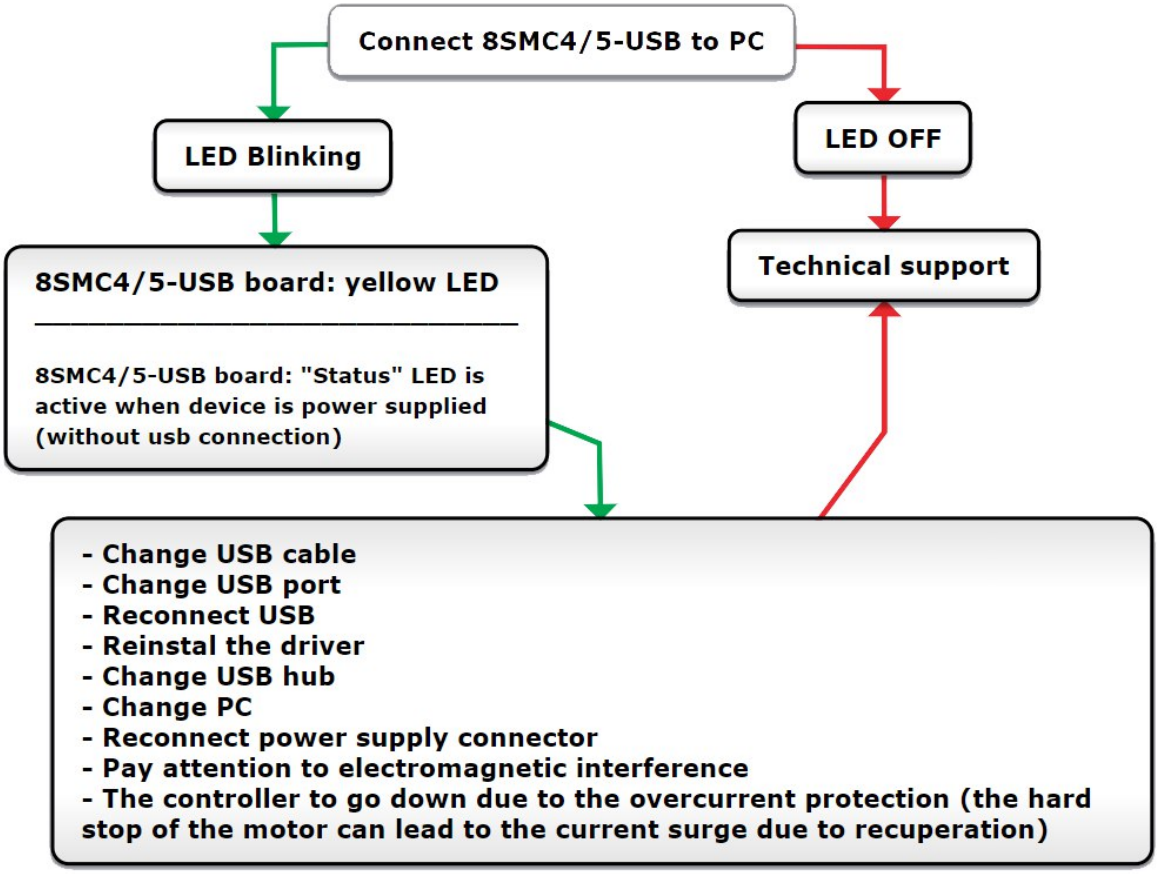
8.1 No device found / Can't open device

- *Connect via USB*
- *Connect via ETHERNET*
  - *If the 8Eth1 adapter is not found on the local network*
  - *The controller serial number is not displayed on the "Common" tab*

8.1.1 Connect via USB

XiLab or other software can not find the controller.

- The PC does not detect the controller via USB:



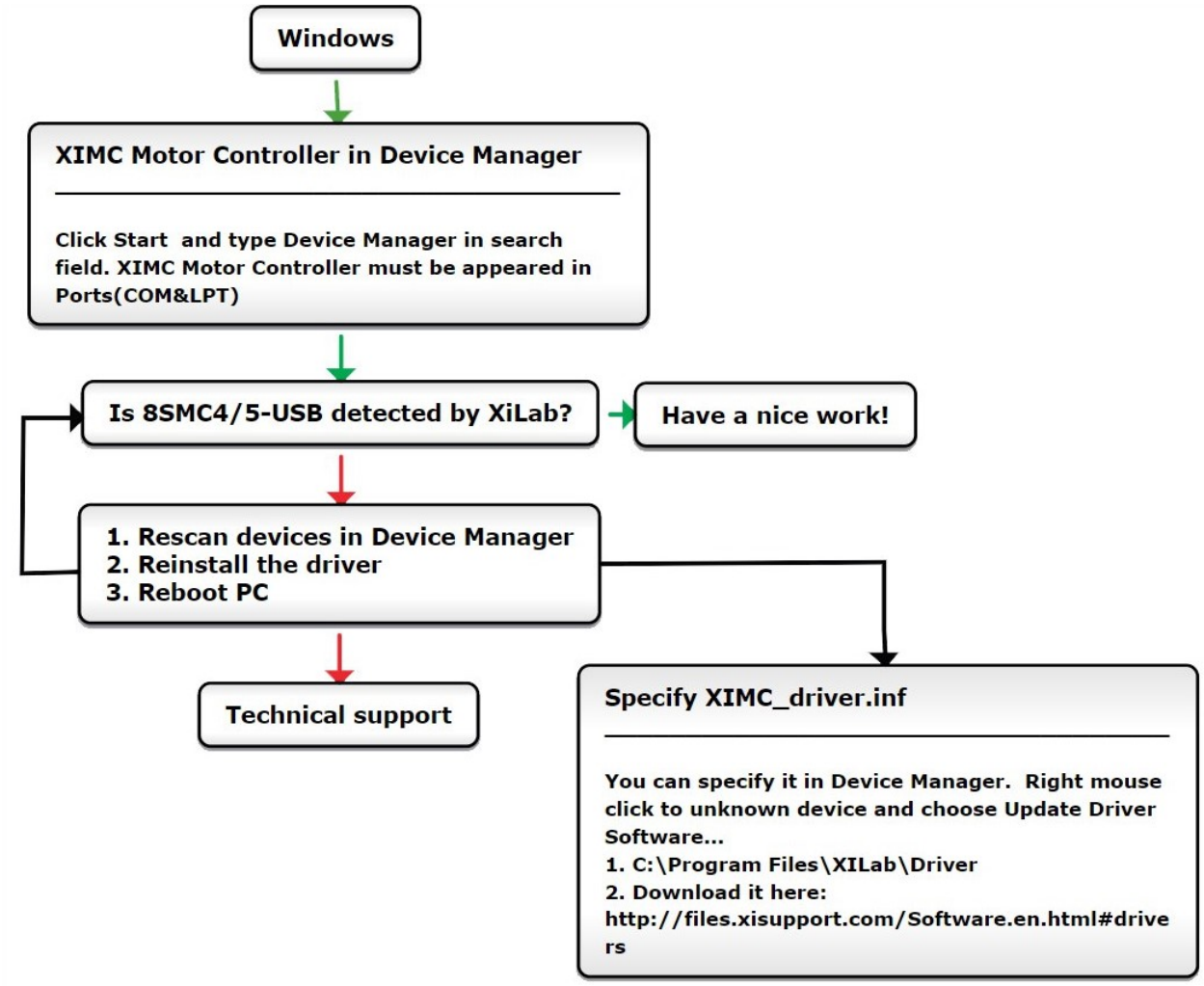
Comment to the scheme:

The most common cause for such type of problem is bad work of the usb-hub, cables or the virtual COM-port identification problem of the operation system on the used PC. Try to reproduce the problem on the another PC or another usb-hub if it is used.

Warning: **"Can't open device" error or "open_device()" function returns -1**. Libximc library opens the controller in exclusive access mode. Any controller opened with libximc (XiLab also uses this library) needs to be closed before it may be used by another process. So at first check that you have closed XiLab or other software dealing with the controller before trying to reopen the controller.

Below are the action maps for the not found controller.

Windows:



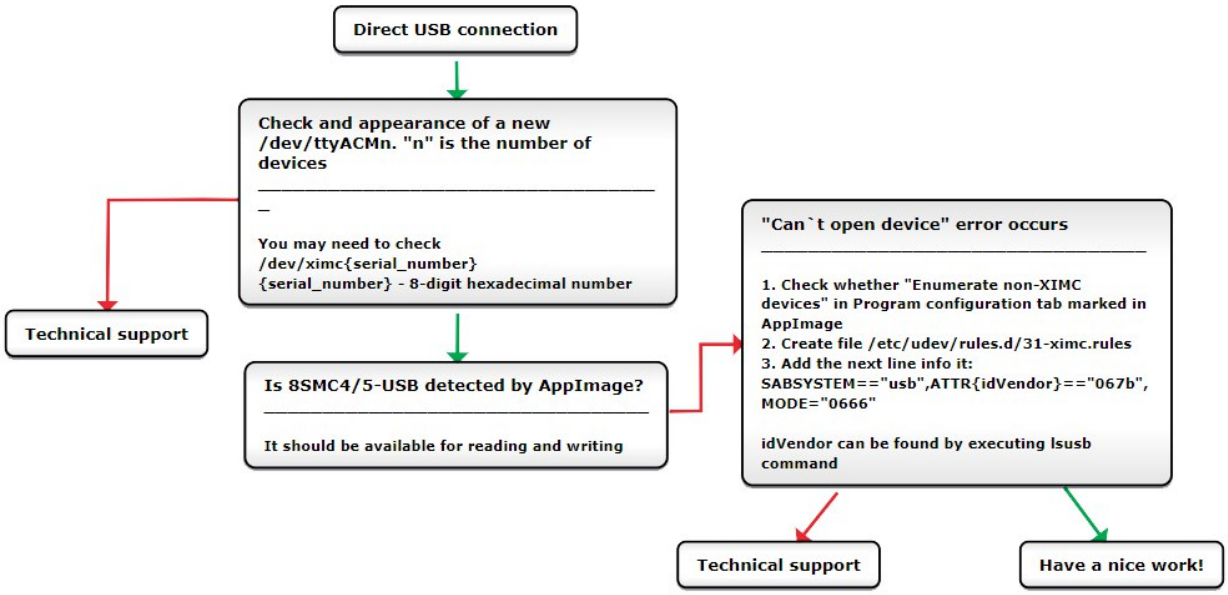
Comments to the scheme:

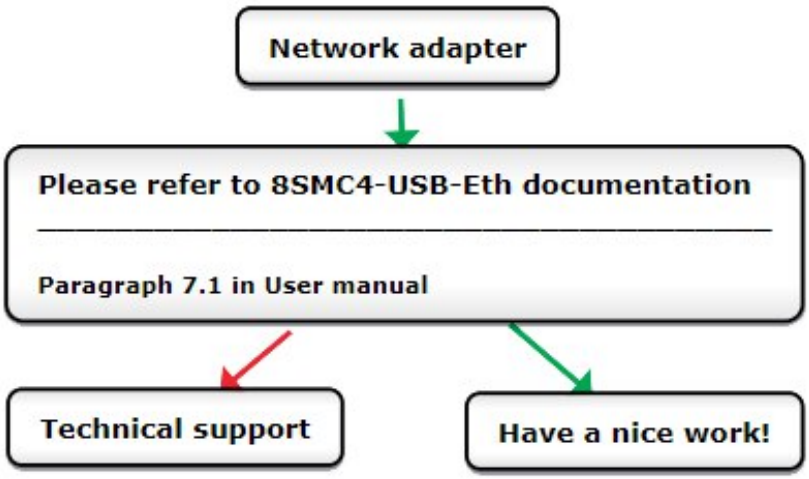
- Check whether the COM-port corresponding to your controller presents in the Device manager. It should be displayed as "XIMC Motor Controller (COMn)". In case the controller has not been recognized, try to reinstall the driver for the controller manually.
- Try to open the COM-port of the controller in any simple serial emulator (Putty for example) and just send the simple command to the controller (like "stop", "sstp", "zero", *"GETS"*, *"GETI"*). The connection parameters are described *here*. The absence of errors means that the controller is operating correctly and the problem should be caused by the used software.

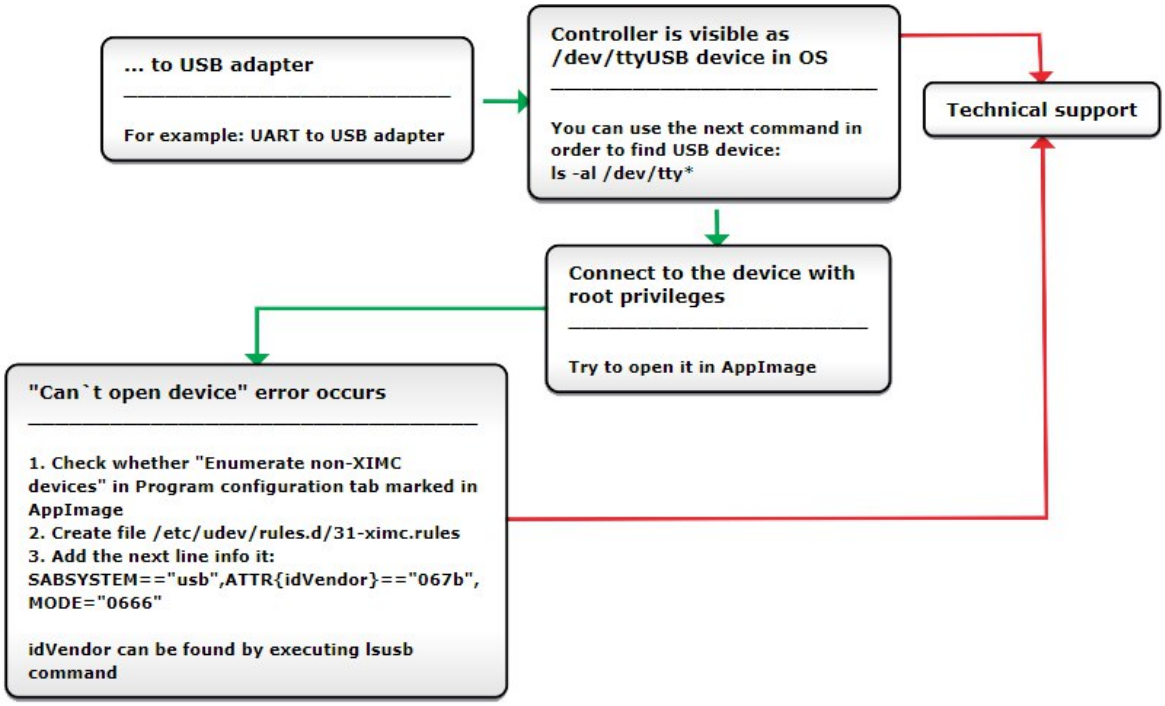
https://youtu.be/HMRpa21B8Dc

Video guide for reinstalling the driver

Linux:


Direct USB connection
Check and appearance of a new /dev/ttyACMn. "n" is the number of devices
You may need to check /dev/ximc{serial_number} {serial_number} - 8-digit hexadecimal number
Technical support
Is 8SMC4/5-USB detected by AppImage?
It should be available for reading and writing
"Can`t open device" error occurs
1. Check whether "Enumerate non-XIMC devices" in Program configuration tab marked in AppImage
2. Create file /etc/udev/rules.d/31-ximc.rules
3. Add the next line info it: SABSYSTEM=="usb",ATTR{idVendor}=="067b", MODE="0666"
idVendor can be found by executing lsusb command
Technical support
Have a nice work!


Network adapter
Please refer to 8SMC4-USB-Eth documentation
Paragraph 7.1 in User manual
Technical support
Have a nice work!



Comment to “Can’t open device” problem solution:

When working with USB-UART converter (as well as USB-Ethernet, USB-Bluetooth etc.) in **Linux** it appears as **/dev/ttyUSB** device. XiLab shows it in a list, but when you try to open it, an error “can’t open device” occurs due to the lack of permissions to the device.

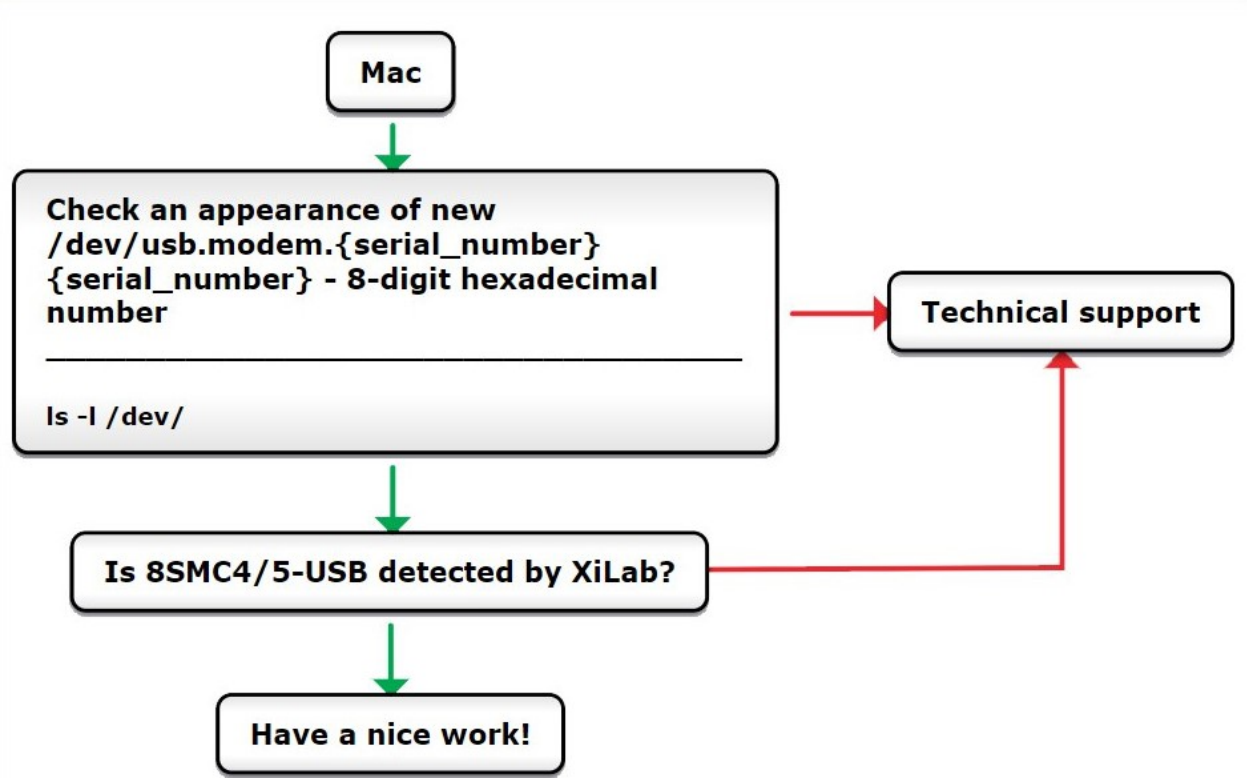
To solve this problem, create a file: /etc/udev/rules.d/31-ximc.rules and add the next line into it:

```
SUBSYSTEM=="usb", ATTR{idVendor}=="067b", MODE="0666"
```

idVendor identifier can be found by executing *lsusb* command.

Note: One possible solution to the “**device not found**” problem is to add a user to the dialuot group. **Important**: after adding it, you must restart your computer.

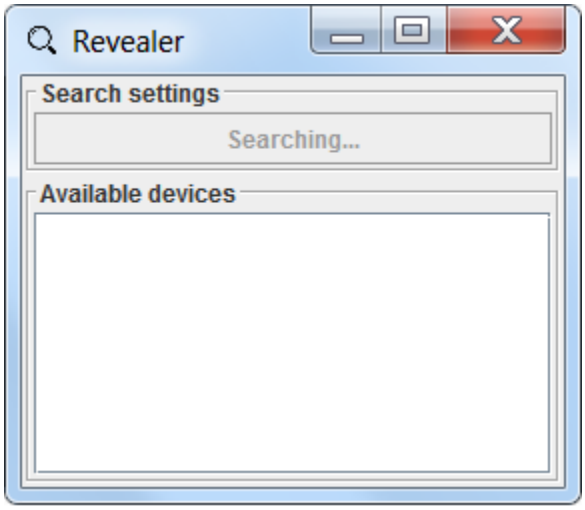
Mac OS:



8.1.2 Connect via ETHERNET

8.1.2.1 If the 8Eth1 adapter is not found on the local network

Note: The **8SMC4-USB-Eth** adapter has been renamed **8Eth1**[1]. All available instructions in our manual are relevant and applicable to both devices.



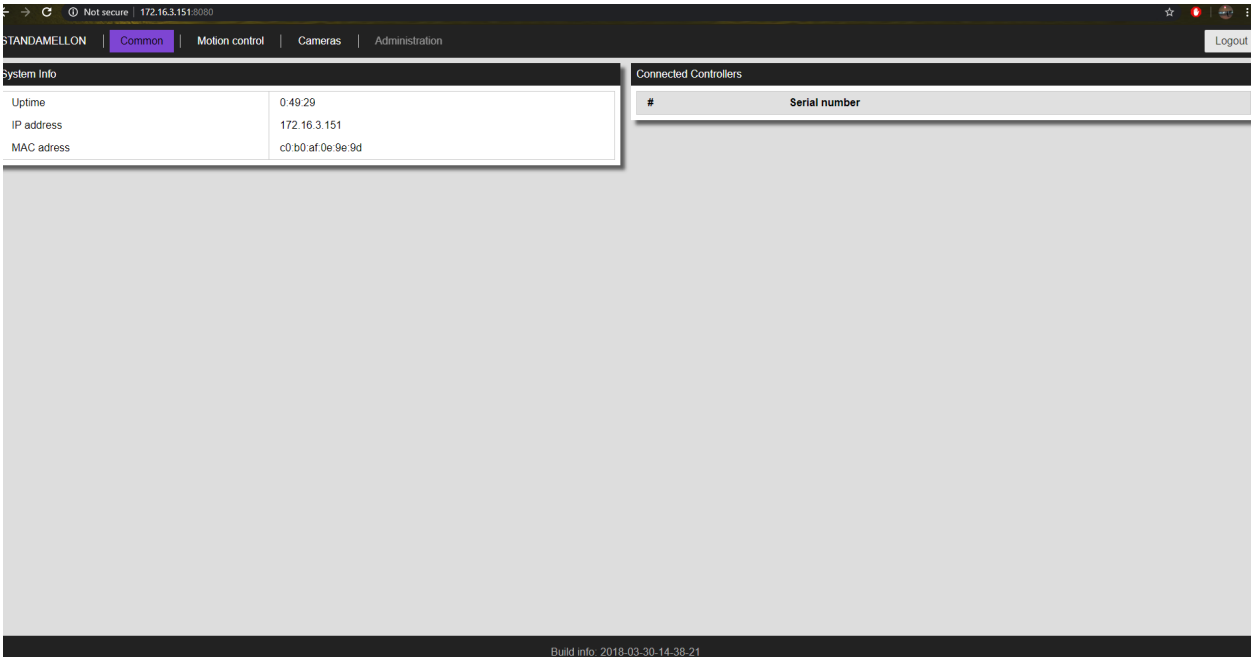
Disable "Windows Defender Firewall" and click the "Search/Restart" button in "Revealer" program. In order to

[1] The vendor code was changed in 2020, earlier this product had the vendor code 8SMC4-USB-Eth.

get access to the Administration panel, navigate your browser to *http://[address]* URL (where *[address]* should be replaced with IP address of the device in your local network and can be obtained with the help of "Revealer" utility). If you are doing this for the first time (or you've disabled cookies/password storage in your browser) **you'll need to authenticate yourself using "admin" as login and password**.

If the control panel opens, your 8Eth1 adapter works fine. **After that, don't forget enable firewall.**

8.1.2.2 The controller serial number is not displayed on the "Common" tab



- Reboot your controller and 8Eth1 adapter
- Connect the controller to your computer via USB and make sure it works correctly. *We recommend using xilab for verification*
- Try to reproduce the problem on another computer

Note: Send your questions with a detailed description of the problem to tech support:

- Ask a question
- Send an e-mail: 8smc4@standa.lt
- Ask a question via Telegram: @SMC5TechSupport

8.2 Unable to rotate the motor by the controller

- *Controller has Alarm state*
- *Motor vibrates without rotation*
- *Mechanical jamming*

- *The motor does not react on move commands*

8.2.1 Controller has Alarm state

Note: Click *Stop* in the main window of XiLab. Controller must return to its normal state.

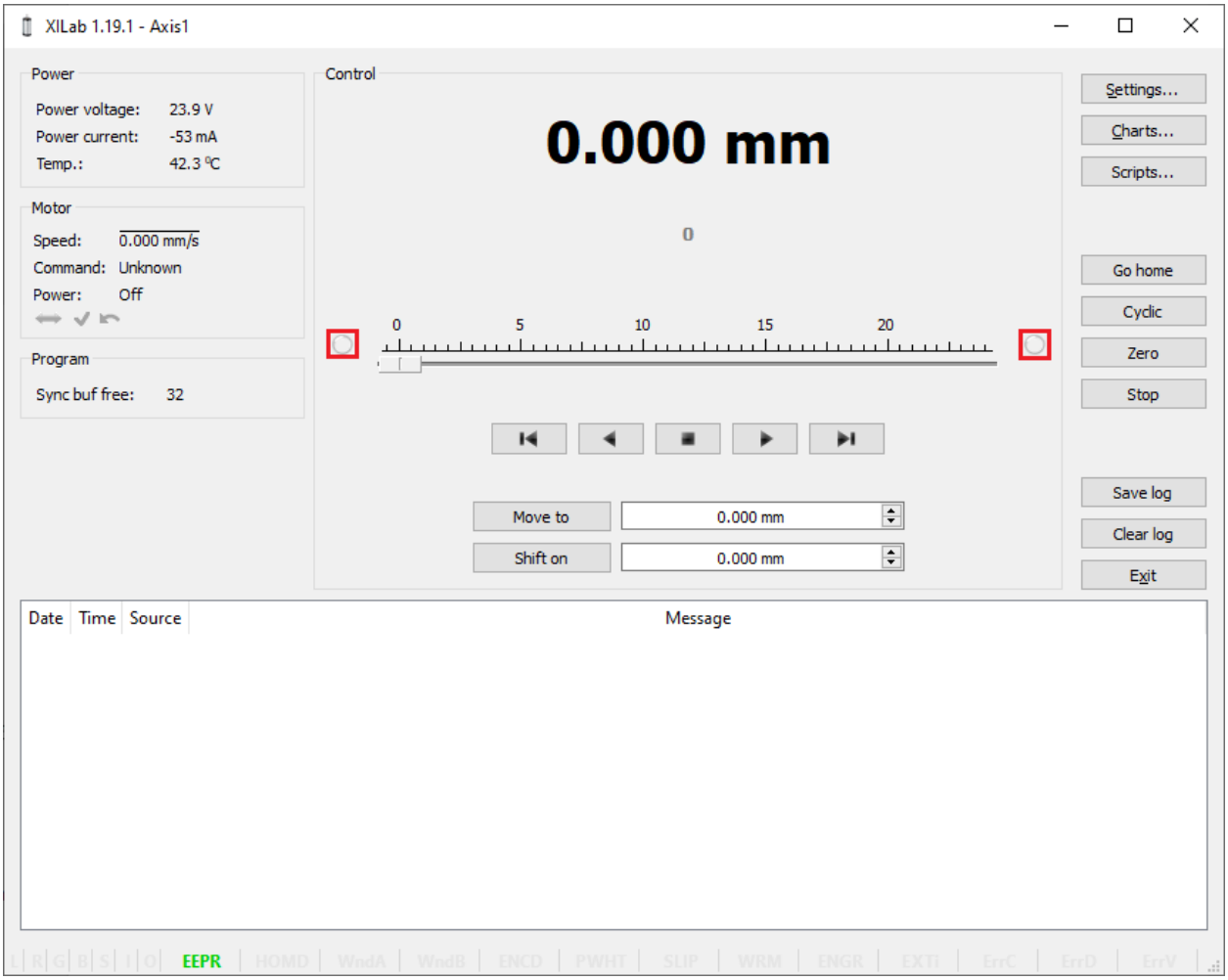
If this approach was not helpful and Alarm state emerged again, do the following:

- Being in XiLab go to the *Maximum ratings* tab.
- Mark *Sticky Alarm flags* option. Click *Ok*.
- Press *Stop* in the main window of XiLab, it will temporary return controller to its normal state. Repeat the sequence of actions leading to Alarm state.
- Make the screenshot of XiLab main window and send it to the technical support with detailed description of your problem.

8.2.2 Motor vibrates without rotation

This problem has several reasons:

- Installed **incorrect profile** for your motor/stage.
  - Search for a better match for the title of profile used by your motorized stage in XiLab folder.
  - It is recommended to save your current configuration to file. To do this, in the *Settings* window of XiLab click *Save settings to file* (see *XiLab settings*), choose path where you want to save the settings. Then send this file in technical support with detailed description of the problem.
- **Incorrect configuration of limit switches**, as result the stage rests the limit switch. This can usually be seen by the indicator lights in XiLab.



The main reason for the incorrect setting of limit switches is incorrect configuration file for your stage (see previous item). Information about manual setting is located in *manual profile setting*. When such problem is emerged it is recommended to contact the technical support for further assistance.

  - It is also one of the consequences of problems with limit switches can be **mechanical jamming** (see the next item).

- **Broken winding** of the motor, problems with bad **contact in connector** etc. It is possible to diagnose this kind of problems independently. For this purpose, we recommend to get XiLab graphs of voltage and current during the operation of motor. The proper motor current in the winding varies according to a sine or cosine. In the broken motor much stronger differences of the current from harmonic form can be noticed.

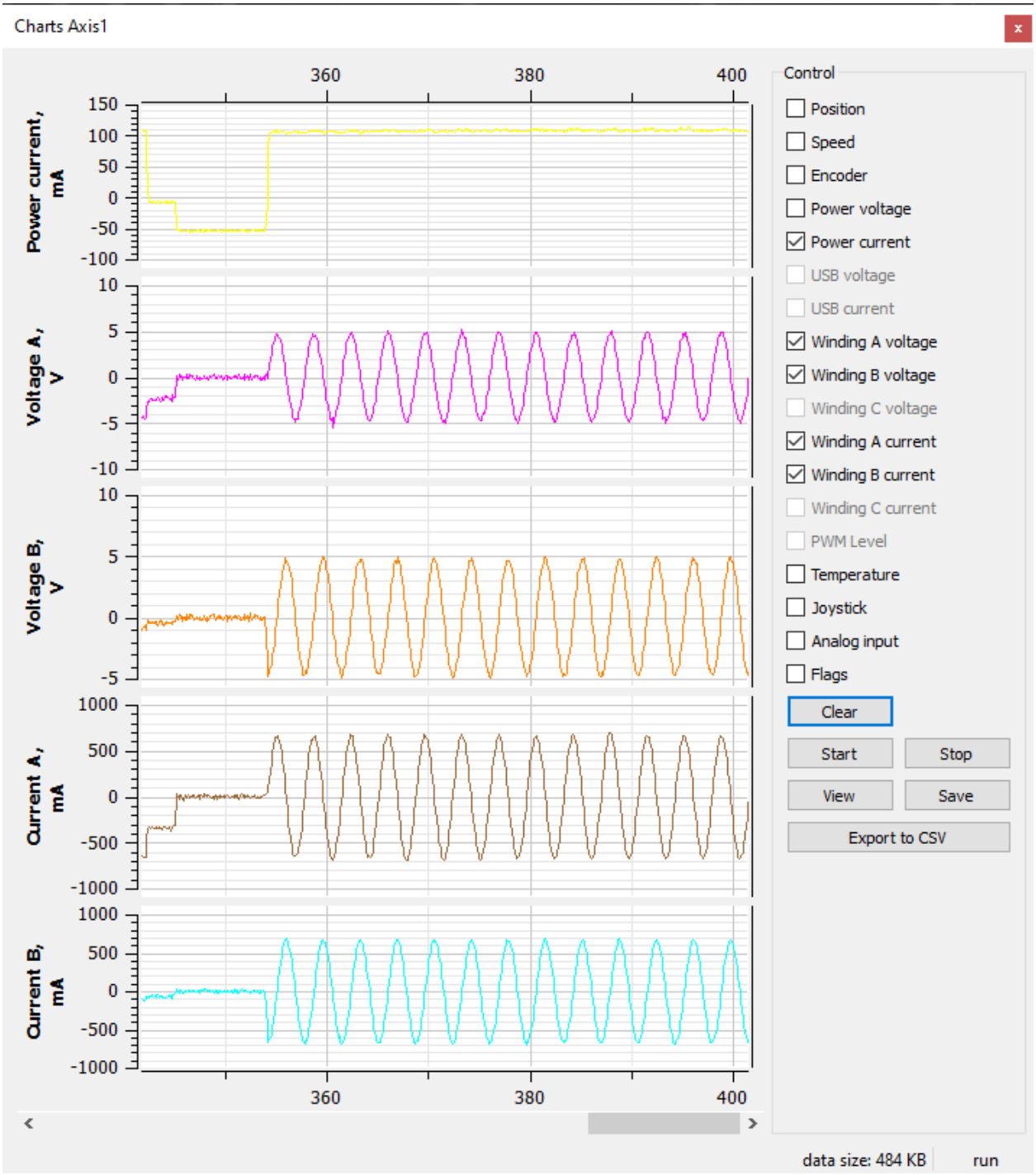


Fig. 8.1: Working case

In the charts below you can see the problems. For example, winding B is open circuit. Probably it is broken. Also, voltage and current forms are distorted.

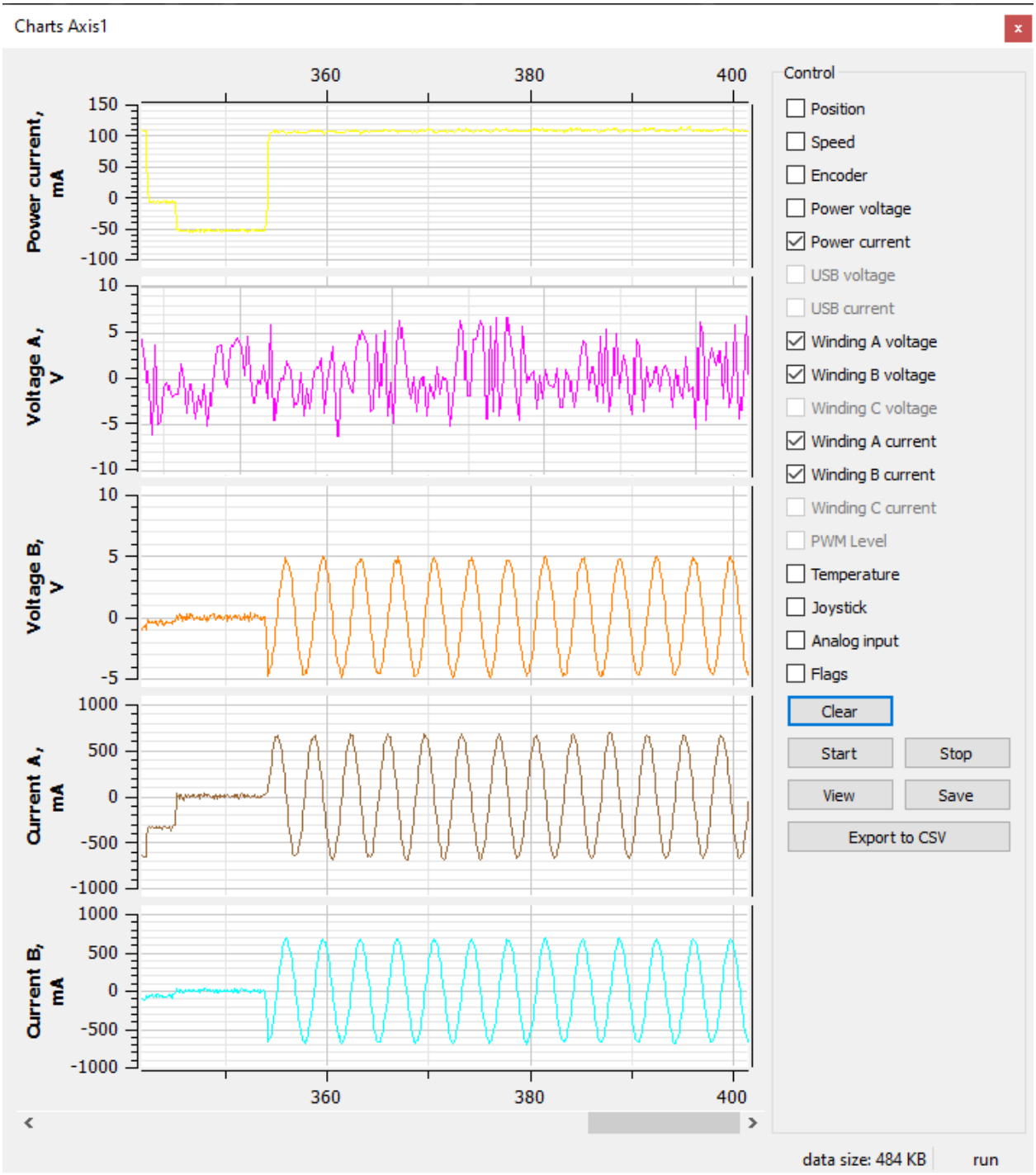


Fig. 8.2: There are problems with motor

To diagnose the problem set very low speed (**1 s/sec** is optimal) and send movement command. Then turn on graphs of current and voltage for windings A and B in XiLab and the Power current (button *Charts*, then mark correspond fields). Wait for a while until the graphs are built. Then it is recommended to send them (Click *Save* in the same window with graphs) in technical support with a detailed description of the problem. Sometimes, when winding is broken, it is impossible to use XiLab due to permanent loss of device. In this case also contact technical support with description of the problem.

8.2.3 Mechanical jamming

There are two ways to deal with jamming:

- Turn the motorized stage by hands if it is possible.
- Increase the winding current 2-3 times for a short time (about 5-10 seconds) and send movement command to the stage in the right direction at the low speed (about **50-100 s/sec** will be enough). A few seconds after rotation, press stop button (small black square) until *power off* status appears in the main window of XiLab in order to prevent motor overheating. **After this do not forget to return the settings back!**

8.2.4 The motor does not react on move commands

The controller looks OK but the motor does not move, leaving error messages in the log, the controller reboots. This bug can arise due to extremely wrong calibration setting of the controller. This happen when the predicted values of the electrical parameters of the motor differ for several orders from the right ones. The wrong calibration sometimes could be caused by the mechanical load on the motor or by differ of mechanical friction in both directions of moving that affects the calibration. So the controller tryes to make a little movement to calibrate the motor (the calibration is performed before the motor power on) and goes down due to the overcurrent protection.

If you encounter this problem just do the following:

- Open XiLab, load the profile for used motor.
- On the *Stepper motor* tab of *Settings* menu change the nominal current to **200 mA**, change the working speed to **1 step/s**, click *Apply* and *Save settings to flash*.
- Try to start moving, watch the current parameters of the motor via *Charts* like it was described above.
- If the charts look OK, then load the normal setting for used motor and work with it as usual.

Note: Send your questions with a detailed description of the problem to tech support:

- Ask a question
- Send an e-mail: 8smc4@standa.lt
- Ask a question via Telegram: @SMC5TechSupport

8.3 Self-test of the 8Eth1 adapter

- *Instructions for installing a static ip image*
- *Static ip address self-test on 8Eth1*

Note: The **8SMC4-USB-Eth** adapter has been renamed **8Eth1**[1]. All available instructions in our manual are relevant and applicable to both devices.

Hint: Software with a static ip address for the 8Eth1 adapter can be downloaded from our website

Instructions for installing a static ip image can be found in the archive and on this page

---

[1] The vendor code was changed in 2020, earlier this product had the vendor code 8SMC4-USB-Eth.

8.3.1 Instructions for installing a static ip image

Attention: Every step of the instructions is mandatory, and no step can be skipped!

-First you must write the image to your microSD card that is in your Ethernet adapter. To do this, remove the microSD card from 8Eth1 and using the card reader connect it to the computer. Using the program *"win32diskimager"* (for Windows) or *"dd"* (for Linux), open the image in the program. The image must be recorded on microSD. -Then you need to remove the microSD card from the card reader and insert it back into 8Eth1. When you start the Ethernet adapter will receive an address via DHCP. -Next you need to connect via SSH on port 36000 (**not 22!**). **login - root**, **password - 12345678**. -Finally, setting a static address. Enter the following at the command prompt:

```
mount -o remount,rw /
vi /etc/network/interfaces
```

-Change static to dynamic, address, netmask, gateway and dns-nameservers. Below is an example. After all the above steps, you need to restart the adapter.

```
auto eth0
iface eth0 inet static
address 172.16.1.101
netmask 255.255.255.0
gateway 172.16.1.1
dns-nameservers 172.16.1.1
:wq
sync
reboot
```

8.3.2 Static ip address self-test on 8Eth1

After the image is written to the SD card, **the dynamic ip address** is set in the configuration file by default. This address is provided to you automatically after you enable the 8Eth1 adapter (automatically assigned a free ip address that is available on your network). If the ip address of the 8Eth1 adapter is not in Revealer after writing the image to the SD card, you need to perform a self-diagnosis:

Important: Self-diagnosis should be performed in an isolated environment. The 8Eth1 adapter must be connected to the computer. **The computer must not be connected to the Internet, and a DHCP server must be running on it!**

1. Make sure that the image was correctly written to the SD card (in the *win32diskimager* program, click the *"Verify Only"* button);
2. Connect the monitor to the HDMI port and the keyboard to the USB port. Make sure that the 8Eth1 adapter is loaded. Wait for the system to boot, the login input field should appear;

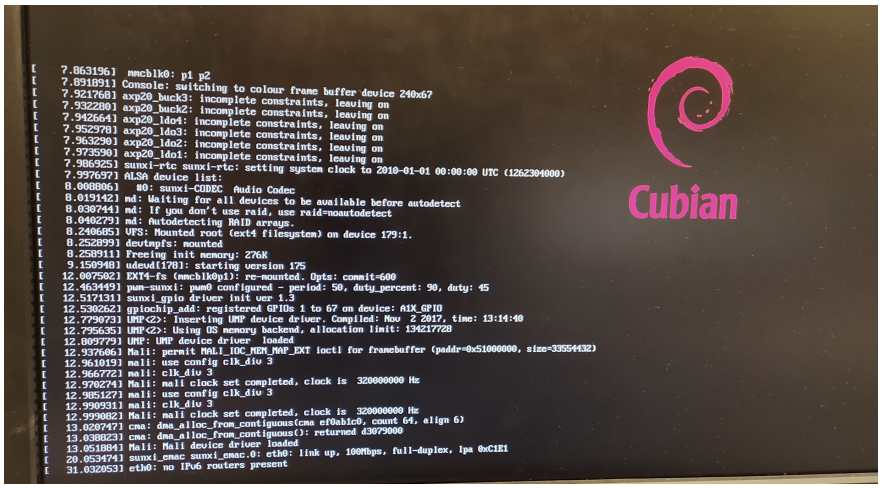


Fig. 8.3: Booting the 8Eth1 adapter system

3. If the system fails to boot, restart the 8Eth1 adapter. At some point, you will see that the 8Eth1 adapter has started. You will see the login input field;
4. Enter **login - root**, **password - 12345678**;
5. Enter **ifconfig** command and make sure that the above adapter has been assigned a dynamic ip address (you need to look at "eth0");

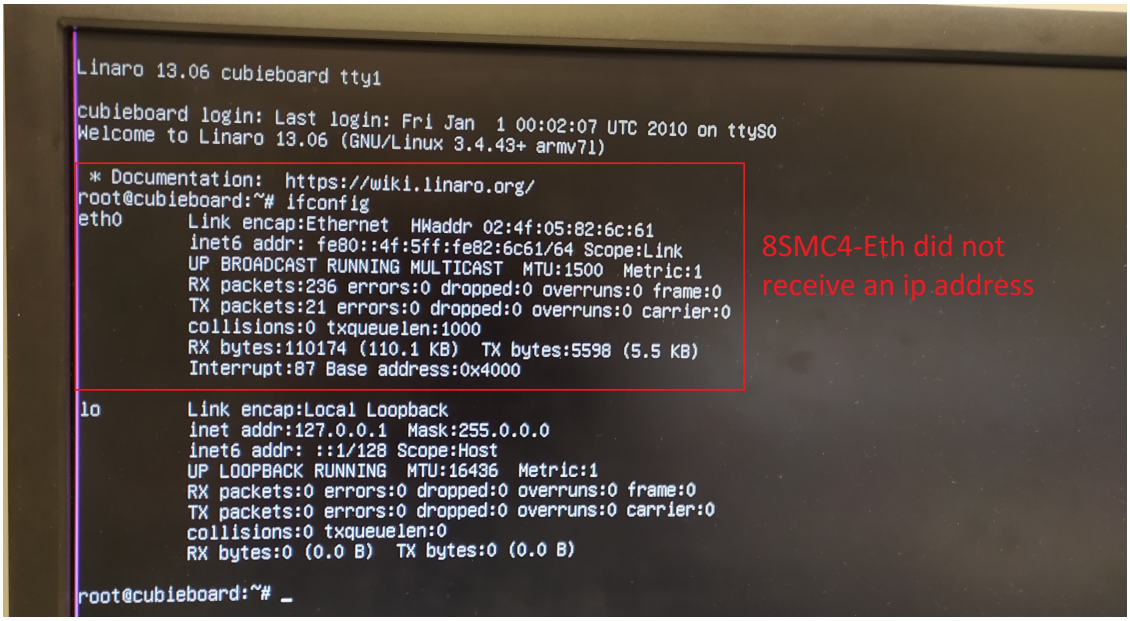


Fig. 8.4: 8Eth1 adapter did not receive an ip address

6. Assign a static ip address to the adapter (the instructions for setting up a static ip address are written above);
7. Disconnect the monitor and keyboard from the 8Eth1. Disconnect the 8Eth1 from the computer and connect it to the network;
8. Don't forget to save the changes and restart the adapter;
9. Wait for the reboot and find the static ip address of your 8Eth1 adapter in the Revealer program.

Note: You can assign a static ip address, bypassing the ssh connection. The steps from the above instructions can be performed immediately on 8Eth1 adapter (you need to make changes to the "interfaces" file)

8.4 When using the libximc library and Linux with kernel version less than 3.16, there are possible hanging of the operating system

Comment: above-mentioned problem stems from the error in serial port driver cdc-acm. It is observed with frequent sequential opening and closing of some devices. Operation system hanging was shown on Debian 7 (3.2 kernel version) and worked correctly on Debian 8 (3.16 kernel version). For additional information about problem please refer to the next link.

Solution: update your current version of Linux.

Note: Send your questions with a detailed description of the problem to tech support:

- Ask a question
- Send an e-mail: 8smc4@standa.lt
- Ask a question via Telegram: @SMC5TechSupport

8.5 USB connection loss

The most common cause of this kind of problem lies in grounding. To find out the reason for the permanent loss of the USB connection, you should:

- If the controller and/or stage is fasten to a metal table, temporarily put something dielectric under it or transfer it to a dialectric surface;
- Ground the computer;
- Ground the controller;
- Ground stage;

Note: If the above steps eliminated the problem of losing the USB connection, then the problem was the grounding of your metal table. There were fluctuations of electrical potential on it.

- Change USB cable. *Use verified USB cables only!* Damaged or low-quality USB cable may cause improper controller operation, including motor rotation errors and errors of device recognition by PC operating system. **Super short cables with thick wires and screening** are ideal for sustainable connection;
- Change USB port;
- Change PC.

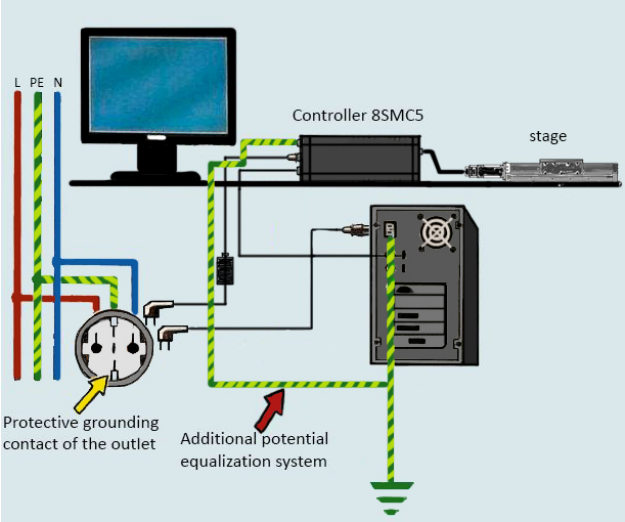


Fig. 8.5: Example of correct grounding

Note: Send your questions with a detailed description of the problem to tech support:

- Ask a question
- Send an e-mail: 8smc4@standa.lt
- Ask a question via Telegram: @SMC5TechSupport

8.6 *probe_flag* - what is it?

```
probe_flags = 1 + 4; % ENUMERATE_PROBE and ENUMERATE_NETWORK
```

"probe_flag" is a parameter passed to the libximc library *"enumerate_devices"* function. It controls how *libximc.dll* does search for devices.

```
define ENUMERATE PROBE 0x01
```

Check if a device with OS name name is XIMC device. **Be carefuly with this flag because it sends some data to the device!**

```
define ENUMERATE NETWORK 0x04
```

Check network devices.

Note: Depending on the type of controller connection, you can remove this or that flag

8.7 Virtual controller as in XILab Software

You can use the virtual controller in your programs. To do this, use the function:

```
device_t XIMC_API open_device (const char *uri)
```

Open a device with OS uri and return identifier of the device which can be used in calls.

Parameters uri- Device uri has form *"xi-com:port"* or *"xi-net://host/serial"* or *"xi-emu:///file"*. In case of USB-COM port the *"port"* is the OS device uri.

For example: *"xi-com:.COM3"* in Windows or *"xi-com:/dev/tty.s123"* in Linux/Mac.

In case of network device the *"host"* is an IPv4 address or fully qualified domain uri (FQDN), *"serial"* is the device serial number in hexadecimal system.

For example: *"xi-net://192.168.0.1/00001234"* or *"xi-net://hostname.com/89ABCDEF"*.

Important: To open network device you must first call **set_bindy_key**

In case of virtual device the *"file"* is the full filename with device memory state, if it doesn't exist then it is initialized with default values. **For example:** *"xi-emu:///C:/dir/file.bin"* in Windows or *"xi-emu:///home/user/file.bin"* in Linux/Mac.

You can also use a virtual controller from XILab with a loaded profile. To do this, select and open the virtual controller in XILab. After loading the required profile ("*Settings*" -> "*Load setting from file...*" button) or just set the necessary parameters and click "*Apply*". Then the file will be saved to the directory *C:Users"user"AppDataRoamingXILab.conf/V_x*, where x - is the number of the virtual controller.

In your program you can open this virtual controller by specifying the full path to the file. **For example:**

```
dev_name = "xi-emu:///C:\Users\"user"\AppData\Roaming\XILab.conf/V_1";
device = open_device(dev_name);
```



Fig. 8.6: Virtual controller operation example

Attention: The profile loading function is only implemented in the xilab interface. If you need to change the stage settings during code execution, you can use an alternative option. You can upload the profile to the controller flash memory. (open XILab->Settings->Load setting from file...->choose your profile->OK->Apply->Save settings to

flash). Then you can change the settings in your program. As soon as you want to return all the default settings, execute the Command READ

8.8 Python CRC algorithm

Below is an example of our **CRC-16/MODBUS** algorithm, which is written in Python.

```
def crc16(data: bytes):
 crc = 0xffff
 for cur_byte in data:
   crc = crc ^ cur_byte
   for _ in range(8):
      a = crc
      carry_flag = a & 0x0001
      crc = crc >> 1
      if carry_flag == 1:
         crc = crc ^ 0xa001
   return bytes([crc % 256, crc >> 8 % 256])


data = b"\x00\x00\x00\xC8\x00\x00\x00\x00\x00\x00\x00\x00"
crc = crc16(data)

crc_str = " ".join("{:02x}".format(x) for x in crc)
print(crc_str)
```

Note: At this link you can find the CRC-16/MODBUS algorithm written in *C#*, *Java* and *PHP*, found on the open source Internet.

The developer is responsible for executing the code.

8.9 Where can I find the programming manual for the 8smc5 controller?

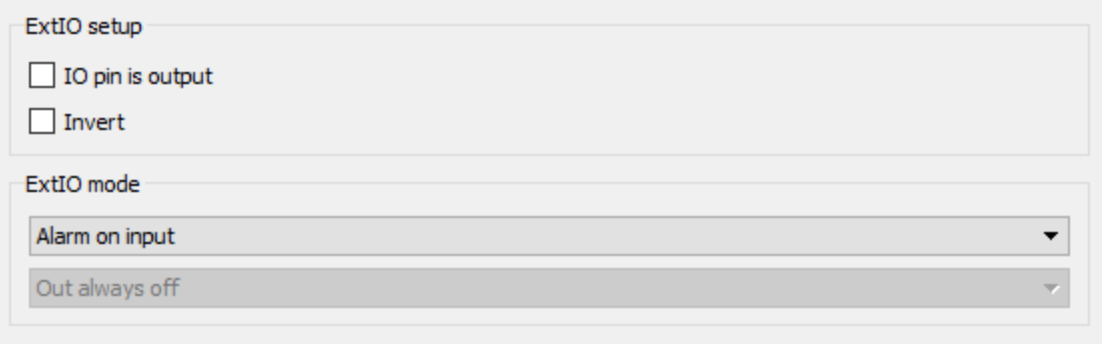
Programming guide is included in libximc 2.X.X archive, where 2.X.X is the version number. It is located in /ximc-2.X.X/ximc/doc-en/libximc7-en.pdf. The programming guide can be found on our second site libximc.xisupport.com or you can download a PDF version of it. The programming guide is Doxygen-based.

8.10 How do I implement an emergency stop button?

To implement an emergency stop button you will need to use a *General purpose digital input/output*, (pins **25. EXTGPIO_0** and **14. DGND, digital ground**) located on the HDB-26 connector.

Using the libximc library, you will need to set the **0x5 - EXTIO_SETUP_MODE_IN_ALARM** flag (see Command SEIO).

If you using XiLab, you need to uncheck the “IO pin is output” in the *EXTIO settiongs* and then select “Alarm on input” from the drop-down list.



Important: It is recommended to use an ALARM for the emergency stop button, because the ALARM will not allow you to perform any actions until it is reset (reset occurs using the stop button or when calling the stop command). If another command is used instead of an ALARM, for example "stop" or "power off", then when any move command (MOVE/MOVR/LEFT/RIGT) is called, the movement will continue, despite the fact that the button is pressed.

The emergency button function does not provide a smooth stop!

8.11 How to get a xilab window that has disappeared off the screen?

All data from installed programs is stored in a hidden *AppData* folder (settings, bookmarks, history, saves, etc). One of the following steps will help you recover your lost xilab window:

Default xilab settings Go to the directory *C:\Users\<your_user>\AppData\Roaming* -> find the *xilab.conf* folder -> rename it or delete it. After that, the xilab settings will be restored to default and all lost windows should return.

You can also manually change the window size. To do this, in the *xilab.conf* folder, find the file with the serial number of your controller, *for example: SM123.cfg*, open it with any text editor and change the fields:

```
[settingsWindow_params]
position = @ Point (411 1057)
size = @ Size (722 286)
```

Arrange the windows in a cascade Click the right mouse button on the taskbar. Select "cascade windows". All open programs will appear in front of you and you can sort them.

Enable display detection Right - click on the desktop and select "screen settings". Then click the Open button. Windows will return the missing Windows to the screen. This helps if the problem occurs due to the presence of multiple monitors.

Change the screen resolution Right - click on the desktop and select "screen settings". In the window that opens, change the permission to another one that is available to you. Windows will move all Windows that go off the screen back to the display. After that, you can return the permission that you had by default.

Use the keyboard shortcut First of all, make the "escaped" program window active. In other words, select it with the mouse on the taskbar, or use Alt+Tab to switch to it. Press Alt+space. It opens a special system menu of the active window. Then click the down arrow on your keyboard and select the second item — **Move**. Press Enter. Now, after pressing Enter, the window is ready to move. Press the left or right key on your keyboard and start moving the window. Continue holding the arrow key until the entire outline is on the visible desktop. Then press Enter.

8.12 How to check if the connection to 8SMC5-USB is established and still active during my session using the libximc library?

To constantly check for a connection between the controller and the libximc library, send the get_status command at regular intervals in the loop.

```
result_t XIMC_API get_status (device_t id, status_t *status)
```

The method described above can be implemented in any program that uses the libximc library.

Note: A similar connection verification method is implemented in XILab

---

8.13 The problem of compensation for backlash (example from technical support)

Note: In this Chapter, we will clearly show the problem of backlash and its solution. The description and solution is taken from the actual user request to the technical support service

---

Dear Service Engineer, We are using one of your motorised mirrors (8MKVDOM-1) with the 8SMC5-USB controller in a very sensitive spot to align our laser. To align our system we observe the power at the output of our laser system. The problem we a re facing is that we get different results when scanning in one direction and go backwards again, i.e. the 'optimal' position changes. We observe this when using our homemade Labview program as well as the XILab software. Using the 'Go home' command in the XILab software does not make the motor go to position 0, but results in an oscillating motion, stopping somewhere near Zero. Is this intentional and does it have anything to do with our problem? The stop setting is using the limit switch, we have not tried to use 'revolution sensor' or 'synchronization pulse'. Is that an option? Also do you have any suggestions how to make the motor as precise as possible?

Best regards, Marvin

https://youtu.be/yBFrdxXr5ok

Video demonstrating the backlash problem

Hello dear Marvin, First of all, set up a profile for your stage. Open XILab -> Settings -> "Load setting from file…" button -> open "Standa" folder -> chose 8MKVDOM-1.cfg. Don't forget to click "Apply" and "Save to flash" buttons. After that click the "Go home" button in the xilab main window and click "Zero" button. Revolution sensor or encoder will help you better control position

In your mechanical system, the moving to the desired position on the left and right is not the same and there is a backlash, but it is possible to eliminate this problem. To do this, check the box next to "Backlash compensation" in the "Stepper motor" tab and specify a value that exceeds the amount of backlash (in your case, this is 8-10 steps). The sign of this setting determines the direction of moving to the position. Positive sign means move from the left while negative - from the right. In the "Backlash compensation speed" field, set the speed at which the compensation movement will be performed. This value should be low (50 s/sec is enough) in order to avoid "drifting" during play compensation.

Important: The backlash compensation mode presumes no axis position correction, providing only a choice of the direction from where the stage should approach the destination point, sticking to this selected direction.

Best regards, Vlad

Hey Vlad, Awesome! We did some more tinkering and it kind of works now

Thanks! Marvin