Motion Engineering, Inc.

33 South La Patera Lane
Santa Barbara, CA 93117-3214
ph (805) 681-3300
fax (805) 681-3311
tech@motioneng.com
www.motioneng.com

APPLICATION NOTE 206, REV. D

SIM4 Option: Installation and Calibration (DOC. NO. 9D00-0106)

CONTENTS

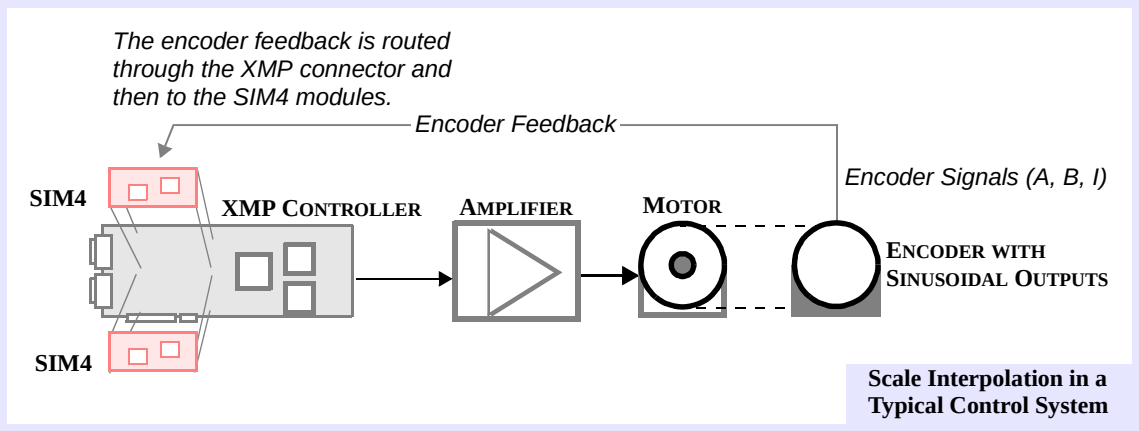
Document Revision History: Application Note 206			
Rev.	**Date**	**Description**	**DCR No.**
D	2001JAN04	Sect. 206.4, "Maximum Count Rate."	525
C	2000APR17	Added steps to the hardware installation chapter	423
B	2000APR12	Updates: 206.6; 206.7.	360

206.1 INTRODUCTION

If your XMP application requires increased position resolution and your scale has sinusoidal outputs, you can use the SIM4 (scale interpolation module) with the XMP. The SIM4 module is available in two configurations: voltage mode or current mode.

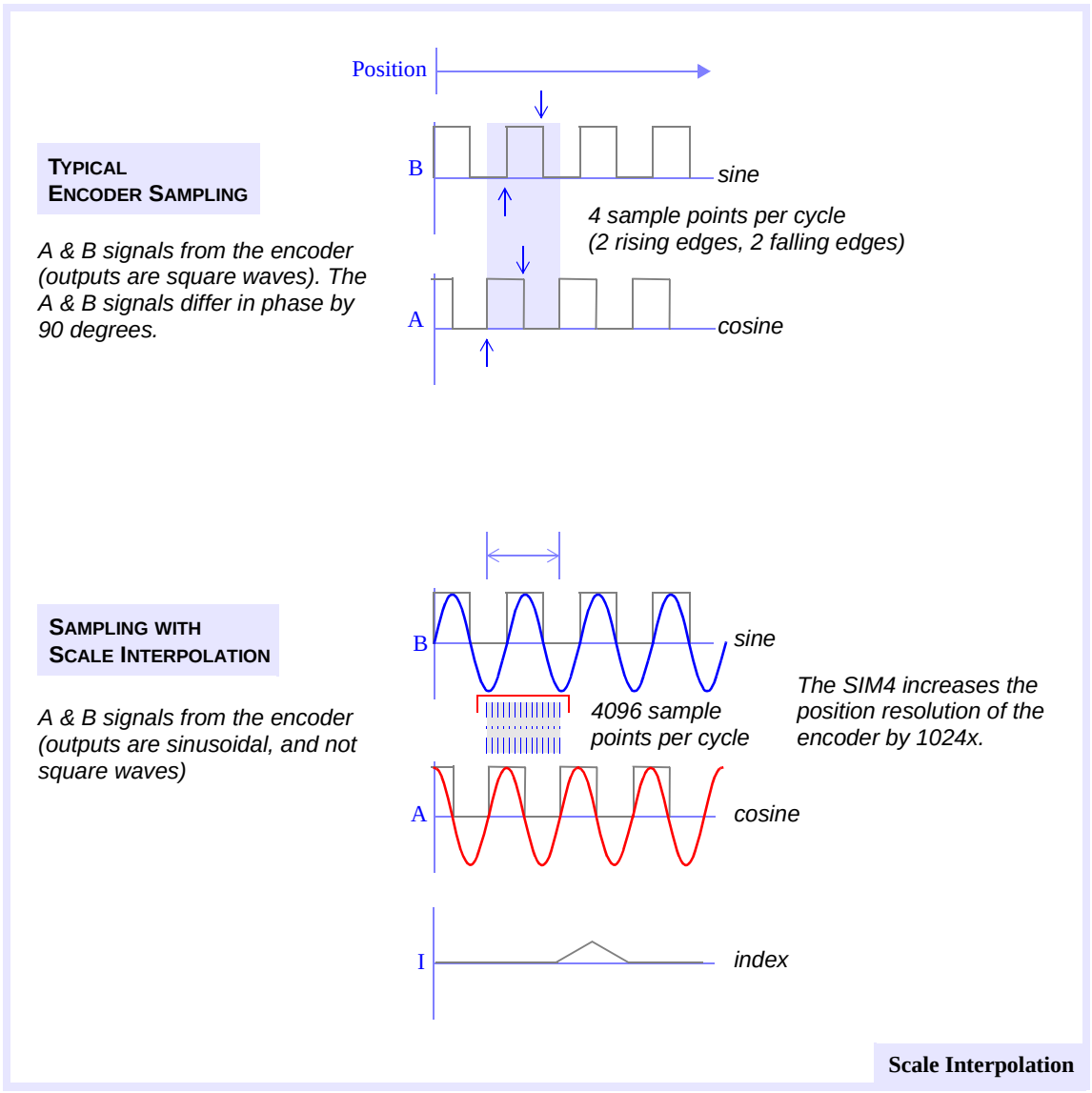
A SIM4 with the *Voltage Mode* option interfaces with a *voltage encoder* output (1 V peak-to-peak differential). A SIM4 with the *Current Mode* option interfaces with a *current encoder* output (11 microamperes peak-to-peak differential).

Figure 1. SIM4 Scale Interpolation in a Typical Control System



206.2 About Scale Interpolation

Figure 2. Scale Interpolation of Encoder Pulses



IMPORTANT: When enabling the SIM4, adjust filter parameters and limits to allow for encoder count rates 1024 times larger than non-interpolated count rates.

206.3 SIM4 ARCHITECTURE

Figure 3. XMP Architecture for 8 Motors

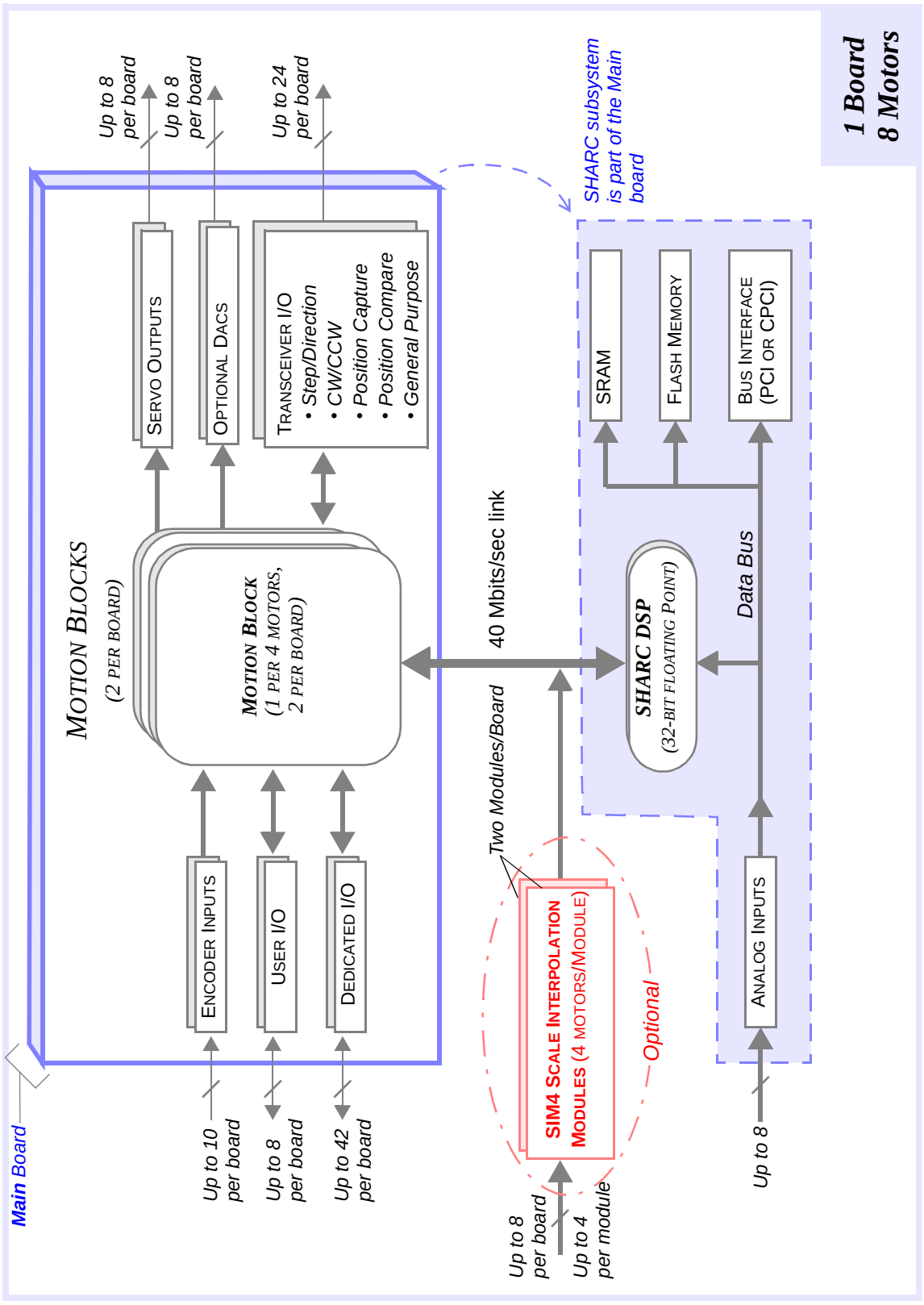
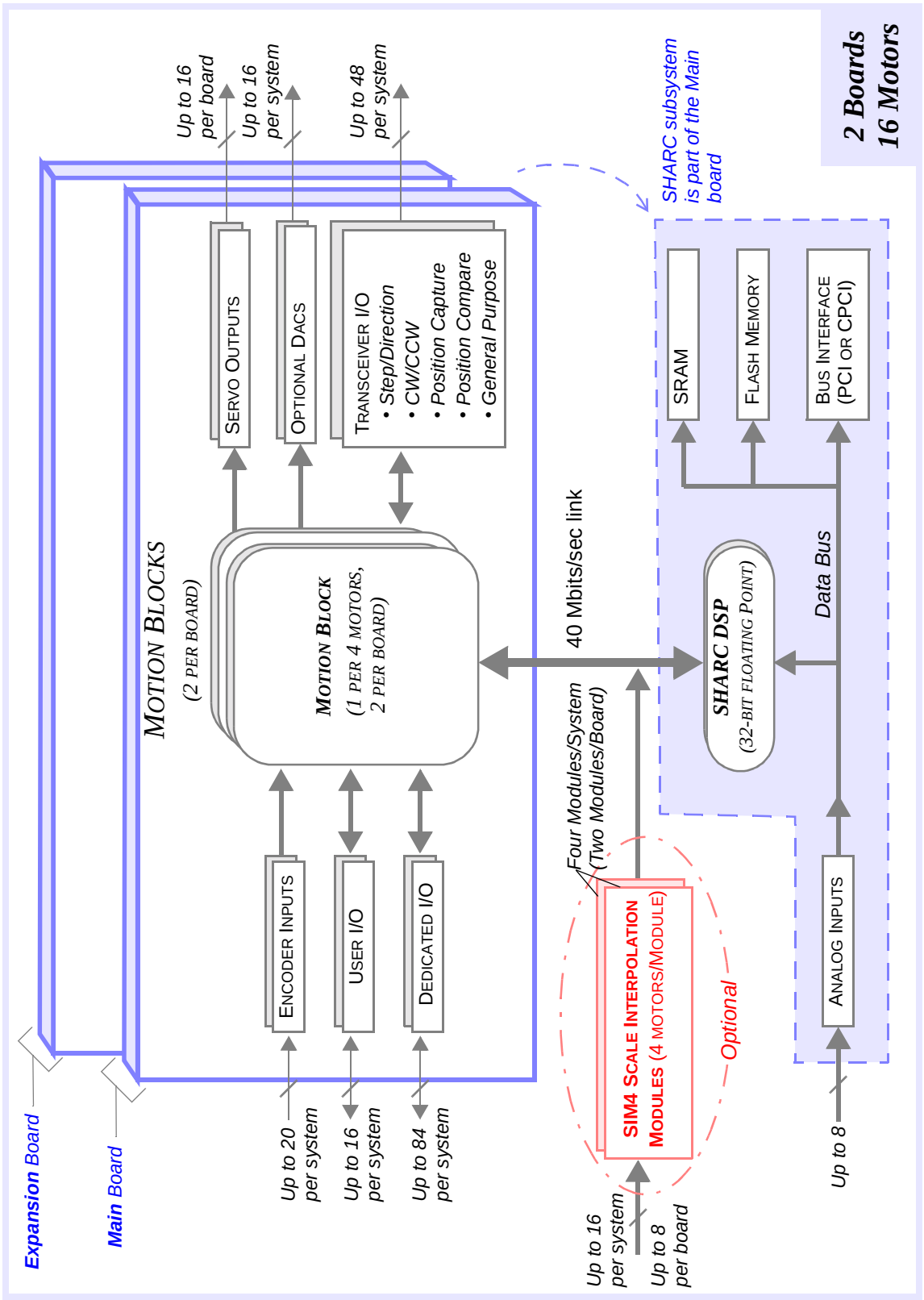


Figure 4. XMP Architecture for 16 Motors

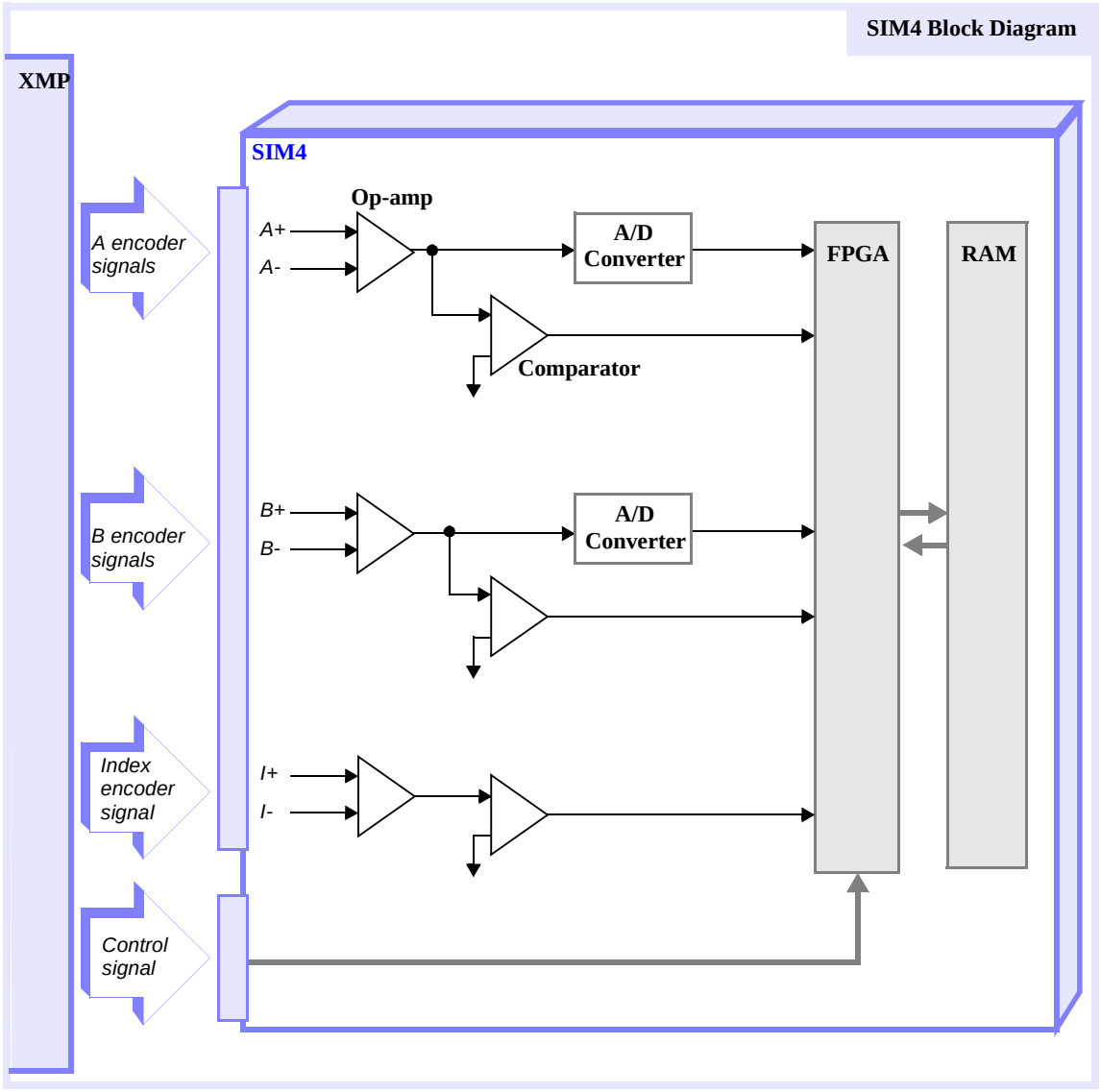


206.4 SIM4 Block Diagram

Connections to the SIM4 module are made through the standard input connectors. The COS (cosine) signals are connected to ENCA (encoder A) inputs, the SIN (sine) signals to the ENCB (encoder B) inputs, and the encoder index signals to the ENCI (encoder index) inputs.

Motors are set up individually in interpolated or non-interpolated mode through the MPI. Capture and Compare are handled in the same way for interpolated as for non-interpolated configurations.

Figure 5. SIM4 Block Diagram



For *encoders with sinusoidal outputs*, the scale interpolation module provides increased position resolution for up to 4 motors per SIM4. These outputs produce 1 cycle of sine and cosine analog signals for each scale *line pair*. At every sample, the scale interpolation module reads the sine and cosine levels and determines the angular position within the encoder's line pair. The sine and cosine outputs are also routed to the standard quadrature inputs, providing coarse position information.

The quadrature inputs generate 4 counts for every line pair, while the 12-bit interpolated value generates 4096 counts for each line pair. The full interpolated position is obtained by combining the *number of quadrant counts* divided by 4, with the position between two encoder lines. The 12-bit scale interpolation provides a resolution increase of 1024 over the quadrature counters. To maintain accurate phase information, the sine and cosine signals are captured with simultaneous-sampling A/D converters.

Position Compare

To implement the position compare feature, the SIM4 compares the current position to a position latched in the FPGA. The A/D converters convert continuously, with a 10 µsec latency.

- 10 compare registers are available per SIM4 (4 motors)
- Can compare 2 positions per servo cycle on 4 motors, or can compare 10 positions per servo cycle on 1 motor
- Delay from compare event to compare = 10 µsec

Maximum Count Rate

The maximum count rate of the SIM4 is determined by the rolloff of the analog input stage of the SIM4. The 3 db point is at 200 kHz. The SIM4 will operate at higher frequencies, but since the signal will be attenuated more at higher frequencies, the signal-to-noise ratio will go down. The highest frequency of operation will depend on possible DC offsets in the raw signals.

Running the SIM4 at sin/cos frequencies above 200 kHz is not recommended. Since each cycle produces 4096 counts, the recommended effective upper count rate is 120,000 x 4096 counts/sec.

Position Capture

To implement the position capture feature, the SIM4 latches the full interpolated position when the user-supplied latch signal is pulsed. Note that different motors can be latched independently of each other.

- 10 capture registers are available per SIM4 (4 motors)
- Can capture 2 times per servo cycle on 4 motors, or can capture 10 times per servo cycle on 1 motor
- Delay from capture event to capture = 5 µsec

206.5 SIM4 HARDWARE INSTALLATION

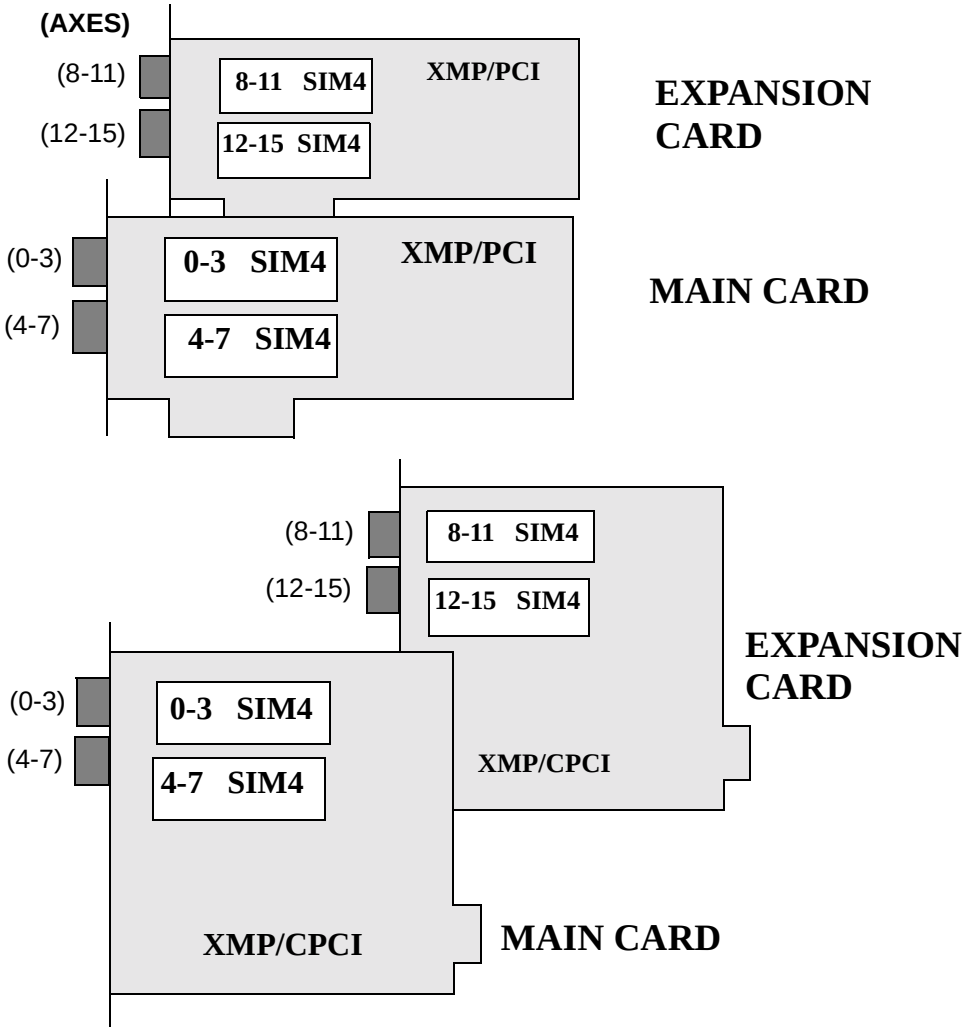
The SIM4 module is a mezzanine-style module designed to plug directly into the XMP control card; up to two SIM4 modules may be installed per board. A simple procedure is used to install a module.

NOTE: For steps #2-5 below, observe all electrostatic discharge (ESD) precautions, including use of a grounded ESD wristband and component mat. Failure to observe ESD precautions may result in damage to the SIM4 module and/or the XMP controller.

1. Power down all motion control equipment and the computer. Unplug all motion control components from the controller.

2. Remove the XMP controller from your computer. Remove the SIM4 module from its protective ESD bag, along with the two m2.5 x 6mm panhead screws, and standoff.

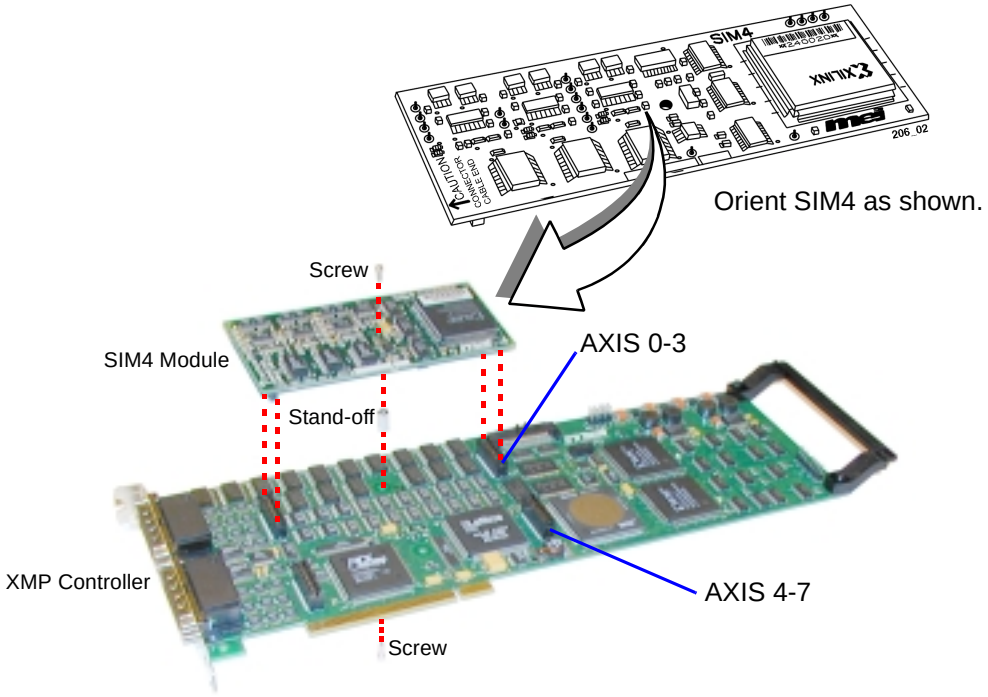
3. Depending on which axes use interpolation, one or both SIM4 modules will need to be installed on the XMP Main Card. Up to 2 more additional SIM4 modules may be added to the Expansion Card if necessary. Interpolation will only occur if the SIM4 modules are connected to the corresponding axes. *See sketch below.*



4. Refer to the figure below for connection procedure.

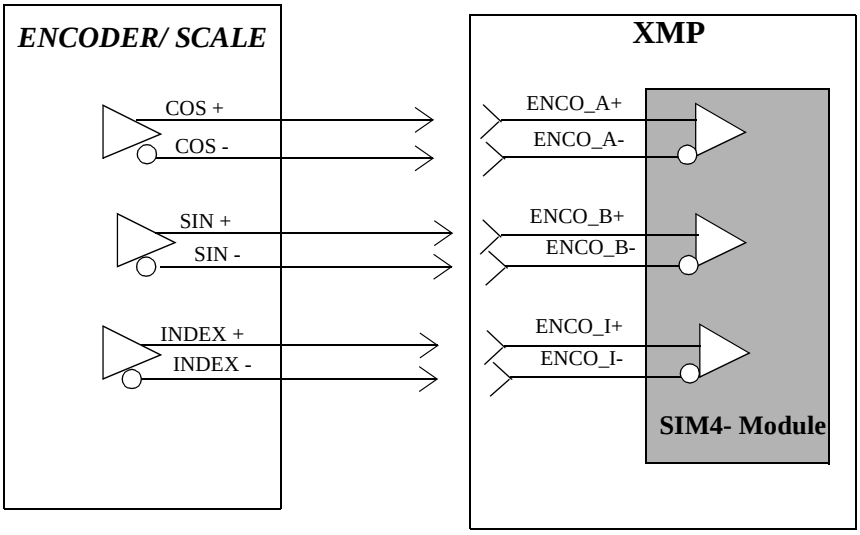
Insert one m2.5 x 6mm panhead screw through the hole in the middle of the SIM4 board, then secure the hex stand-off to the **underside** of the SIM4 module. Do not overtighten screw! Align the SIM4 module with the XMP controller as shown in the figure below, and press the module into a pair of receptacles. (There is one molded connector plug at each end of the SIM4 which mates with a corresponding receptacle on the controller.) The SIM4 module **must** be oriented as shown in the figure to function properly.

Insert a second m2.5 x 6mm panhead screw through the hole in the underside of the XMP controller, then secure it to the stand-off between the XMP controller and SIM4 module.



5. Reinstall the XMP controller with the SIM4 module into the computer. Connect all cables and power up the system.

6. When wiring the Encoder (or Scale), the signals connect directly to the same pins as a normal digital encoder. Connect the analog scale outputs into the standard quadrature inputs of the XMP. See the connection diagram below for an example of how to wire up the encoder (or scale). Refer to *XMP Hardware Installation Manual* for detailed connector pinouts.



206.6 SIM4 CONFIGURATION

To operate the XMP with scale interpolation, the SIM4 module is installed onto the XMP controller (see Section 206.5 above), and the XMP is configured for scale interpolation. Configuring the XMP for SIM4 operation is illustrated in the sample application Sim4.c.

The XMP can also be set up for SIM4 operation through Motion Console, by toggling the SIM4 button in the **Motor Summary / General Config** screen.

After executing Sim4.c, the XMP returns position measurements with a resolution of 1024 times the quadrature resolution. The limits and filter parameters must therefore be adjusted to correspond to the higher count rate.

Once these steps are taken, the XMP is ready to operate in interpolated mode.

In interpolated mode, the Xmp/SIM4 determines the angular position at a given time by sampling the sine and cosine signals simultaneously, and looking up the angle in the arctan lookup table, located in the SIM4 flash. Ideally, the sine and cosine signals should have no DC offsets and should have equal amplitudes. If there are DC offsets and amplitude differences, the angle obtained from the lookup table based on the raw values will be incorrect. Under these conditions, increased accuracy is obtained by calibrating the sinusoidal scale and compensating the sine and cosine values before looking up the angle in the arctan lookup table as described below.

Compensation is illustrated in the series of figures below. Figure 8a shows the ideal sine/cosine relationship. Figure 8b shows the case where the cosine amplitude is larger than the sine amplitude, and where both sine and cosine have a DC offset. Figure 8c shows the sine and cosine after compensation.

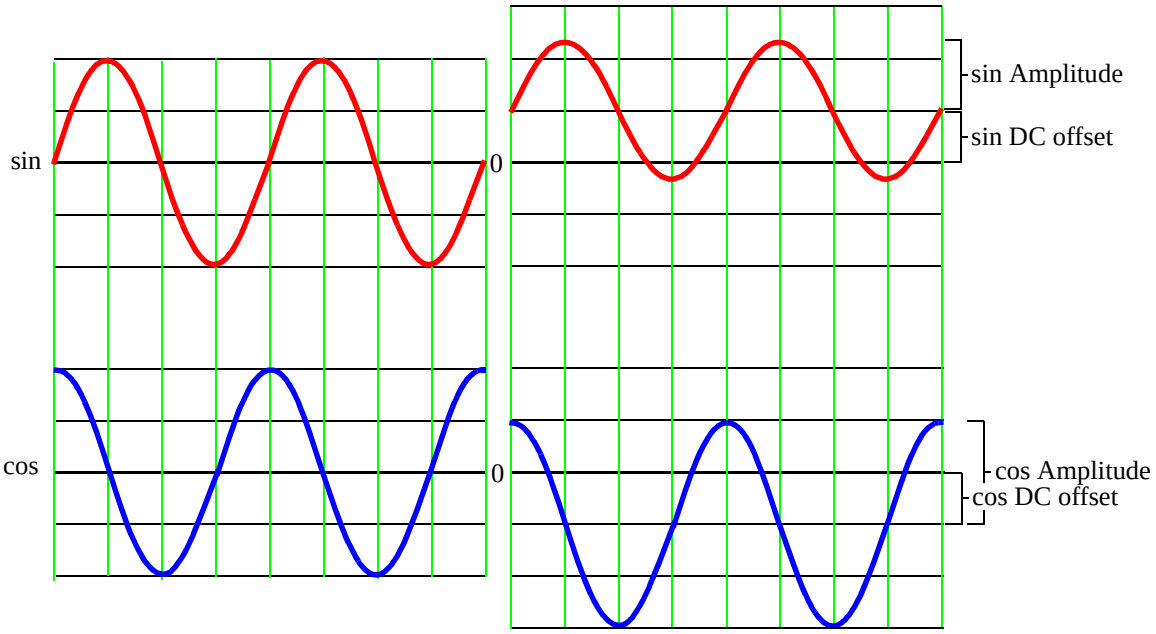
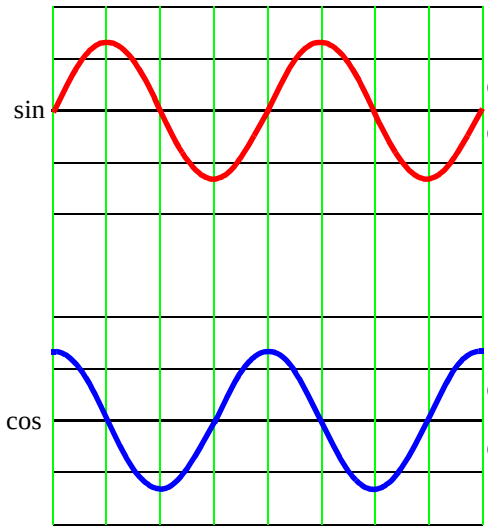


Figure 8a. Ideal sin/cos relationship.

Figure 8b. Imperfect sin/cos relationship before compensation.

Figure 8c. Imperfect sin/cos relationship after compensation.



Compensated sin DC offset = 0

Compensated sin Amp = sin Amp

Compensated cos DC offset = 0

$$\text{Compensated cos Amp} = \text{cos Amp} \times \frac{\text{sin Amp}}{\text{cos Amp}}$$

NOTE: if sin Amp > cos Amp

$$\text{Compensated sin Amp} = \text{sin Amp} \times \frac{\text{cos Amp}}{\text{sin Amp}}$$

Compensated cos Amp = cos Amp

The first part of the calibration process for a given motor consists of measuring the DC offsets and the amplitude differences, then computing a set of compensation lookup tables. This is done with Sim4Cal.c. This application moves the scale over a short distance in the region of operation and determines the maximum and minimum of each signal. One sine/cosine sample is collected every servo cycle. The move distance should be long enough to encompass several cycles of the encoder signals, and the velocity of the move should be low enough to sample at least 50 points per sine or cosine cycle. The compensation tables for sine and cosine are stored in an output file.

The second part of the calibration consists of loading the compensation tables into the flash on the sim4 module. This is done with Sim4Flsh.c.

The contents of the SIM4 flash are non-volatile. They will remain unchanged through resets and power-downs. A compensation table can be replaced by an improved one or by a unity-transfer-function table by a new execution of Sim4Flsh.exe with the appropriate arguments. The unity-transfer-function compensation table is generated by running Sim4Cal with the “nocal” argument.

If desired, the relevant sections of Sim4Cal.c and Sim4Flsh.c can be combined within the user’s application code to perform scale calibration in-line.

The details of the programs follow.

206.7 SAMPLE APPLICATIONS

The sample application programs below can be used to configure and calibrate your SIM4 module. All of them are found in the software tree shipped with your hardware. The **Sim4.c** sample application is detailed in the *MPI/XMP Sample Applications Manual* (P/N M001-0065).

- **Sim4.c** Sets up the XMP for SIM4 operation.
- **Sim4Cal.c** Obtains a compensation table for a single motor.
- **Sim4Flsh.c** Loads the arctan lookup tables and motor compensation tables in the SIM4 flash.

Sim4.c

This sample application configures a motor for SIM4 operation and for capture on the Encoder Index ANDed with ENCA and ENCB (included at end of this note).

Sim4Cal.c

This sample application computes the compensation tables for a SIM4 motor and saves the data in a file.

The program assumes that the PID parameters, error limits and velocity tolerances are set up prior to running this application. The program sets the relevant motor up for SIM4 operation, sets up the Data Recorder, initiates a constant velocity move for MAXRECORDS data points, computes the compensation tables, and writes them to the output files in binary and ascii format. The default output file names are cal.abs (binary form of compensation tables), cal.txt (ascii form of compensation tables) and output.txt (ascii form of measured values).

Command line arguments:

- -motor # : specifies the motor
- -file <filename> : specifies the file name
- nocal : generates unity sine and cosine compensation tables

"Sim4Cal" without any arguments generates the compensation tables for sine and cosine for motor 0 in the file cal.abs. An ascii version of the data is stored in cal.txt. The raw values collected for sine and cosine are stored in output.txt.

"Sim4Cal -file <filename>" generates the compensation tables for motor 0 and stores the data in files filename.abs, filename.txt and output.txt.

"Sim4Cal cal -motor # -file <filename> generates the compensation tables for motor # and stores the data in files filename.abs, filename.txt and output.txt.

"Sim4Cal nocal" generates unity transfer sine and cosine compensation lookup tables (i.e. tables applicable for zero DC offsets and equal amplitude for sine and cosine) and stores the data in files cal.abs, cal.txt and output.txt.

Sim4Flsh.c

This application loads the SIM4 flash with the arctan table and the sine and cosine compensation tables. SIM4 uses the compensation tables to correct analog sine and cosine inputs for DC offsets and for amplitude differences between the two signals. It then uses the arctan lookup table to obtain the interpolated position value from the corrected sine and cosine inputs.

MEI distributes a default configuration of the flash (sim4.abs). The compensation tables in the default configuration are unity transfer compensation tables for all motors.

The SIM4 flash file is partitioned into areas for the arctan lookup table and the sine and cosine compensation tables. There is a sine and cosine compensation table for each motor. Sim4flsh can load the entire SIM4 flash area or a compensation table for a single motor.

When Sim4Flsh is invoked without arguments, it expects to load the entire flash area with a file named sim4.abs on Motion Block 0.

To flash another MotionBlock, the –block argument is used. A different file name can be specified with the –file option. A compensation table for a specific motor can be loaded with the cal and -motor arguments. Compensation tables can be created with the sim4cal program.

To create a file for the entire flash area, generate the necessary individual compensation tables with the sim4cal program and load them into flash. Then use “sim4flsh get –block #” to write the entire flash area into a file. The default file name is sim4.abs. Take care not to overwrite the default configuration distributed by MEI.

Command line arguments:

- no arguments : load entire flash of a MotionBlock
- cal : load single compensation table
- -motor # : specifies the motor
- -block# : specifies the MotionBlock to write to or read from
- get : read entire flash of a MotionBlock
- -file <filename> : specifies a file to read from or write to

NOTE: -motor and -block are mutually exclusive: -motor is used in cal operations; -block is used in all other operations.

Examples

"sim4flsh" programs the entire flash for the SIM4 module on MotionBlock 0 with the default file "sim4.abs".

"sim4flsh -block # " programs the entire flash for the SIM4 module on MotionBlock # with the default file "sim4.abs".

"sim4flsh cal -motor # -file <filename>" loads a compensation table from the file “filename” for motor #.

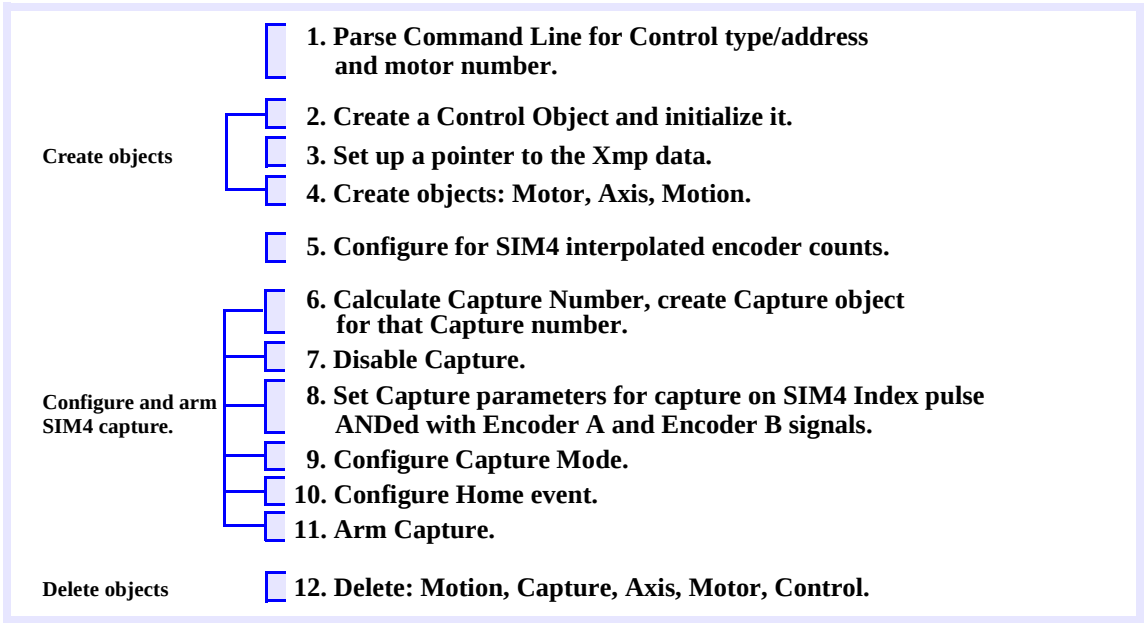
"sim4flsh get" reads the entire flash on MotionBlock 0 and saves the data in the default file "sim4.abs".

"sim4flsh get –block # -file <filename>" reads the entire flash for the SIM4 module on MotionBlock # and saves the data in the file "filename".

WARNING: Sim4Flsh resets the controller.

Sample Application sim4.c

This section contains the sample application sim4.c.




```
/* sim4.c */

/* Copyright(c) 1991-1996 by Motion Engineering, Inc.  All rights reserved.
 *
 * This software  contains proprietary and  confidential information  of
 * Motion Engineering Inc., and its suppliers.  Except as may be set forth
 * in the license agreement under which  this software is supplied, use,
 * disclosure, or  reproduction is prohibited without the prior express
 * written consent of Motion Engineering, Inc.
 */

#if defined(MEI_RCS)
static const char MEIAppRCS[] =
      "$Header: /MainTree/XMPLib/XMP/app/sim4.c 6     2/16/00 4:29p Anns $";
#endif

/*
:Enable XMP SIM4 module and capture on [Index & ENCA & ENCB].

This sample application configures a motor for SIM4 operation and
capture on the Encoder Index ANDed with ENCA and ENCB.

The code assumes that motorN is associated with axisN.

Warning!  This is a sample program to assist in the integration of the
 XMP motion controller with your application.  It may not contain all
 of the logic and safety features that your application requires.

*/

#include <stdlib.h>
#include <stdio.h>

#include "stdmpi.h"
#include "stdmei.h"

#include "apputil.h"

#if defined(ARG_MAIN_RENAME)
#definemain sim4Main

argMainRENAME(main, sim4)
#endif
```

```
/* Command line arguments and defaults */
long  motorNumber= 0;

Arg   argList[] = {
      {     "-motor",ArgTypeLONG,&motorNumber,},

      {     NULL,       ArgTypeINVALID,NULL,}
};

/* User Settings */

#define ACTIVE_LOW_EDGE(0)
#define ACTIVE_HIGH_EDGE(1)

int
      main(intargc,
            char *argv[])
{


/* For MotorBlock 0: */
MEIXmpServiceCmdCaptureMode[MEIXmpMAX_StandardMotors] ={
      0x00020441,
      0x0F0F8180,
      0x00020441,
      0x0F0F8382,
      0x00020441,
      0x0F0F8584,
      0x00020441,
      0x0F0F8786,

/* For MotorBlock 1: */
      0x10020441,
      0x0F0F8180,
      0x10020441,
      0x0F0F8382,
      0x10020441,
      0x0F0F8584,
      0x10020441,
      0x0F0F8786,

/* For MotorBlock 2: */
      0x20020441,
      0x0F0F8180,
      0x20020441,
      0x0F0F8382,
      0x20020441,
      0x0F0F8584,
      0x20020441,
      0x0F0F8786,

/* For MotorBlock 3: */
      0x30020441,
      0x0F0F8180,
      0x30020441,
      0x0F0F8382,
      0x30020441,
      0x0F0F8584,
      0x30020441,
      0x0F0F8786,
};



long captureActiveEdge = ACTIVE_HIGH_EDGE;



      long  returnValue;
      long  argIndex;
      long  captureNumber;
      long  blockNumber;
      long  motorInBlock;
```

```
MPIControl       control;
MPIAxis               axis;
MPIMotor         motor;
MEIMotorConfigmotorConfigXmp;
MPIControlTypecontrolType;
MPIControlAddresscontrolAddress;
MEIXmpData       *Xmp;
MEIXmpBufferData*external;
MPICapture       capture;
MPIMotion        motion;
MPIMotorEventConfigeventConfig;
MPICaptureConfigcaptureConfig;
```

```
/* Parse command line for Control type and address */
argIndex =
      argControl(argc,
                  argv,
                  &controlType,
                  &controlAddress);

/* Parse command line for application-specific arguments */
while (argIndex < argc) {
      long  argIndexNew;

      argIndexNew = argSet(argList, argIndex, argc, argv);

      if (argIndexNew <= argIndex) {
            argIndex = argIndexNew;
            break;
      }
      else {
            argIndex = argIndexNew;
      }
}

/* Check for unknown/invalid command line arguments */
if ((argIndex < argc) ||
      (motorNumber >= MEIXmpMAX_StandardMotors)) {
      meiPlatformConsole("usage: %s %s\n"
                         "\t\t[-motor # (0 .. %d)]\n",
                           argv[0],
                           ArgUSAGE,
                           MEIXmpMAX_StandardMotors - 1);
      exit(MPIMessageARG_INVALID);
}
```

1

```
/* Obtain a Control handle. */
control =
      mpiControlCreate(controlType,
                       &controlAddress);
msgCHECK(mpiControlValidate(control));
```

```
/* Initialize the controller */
returnValue = mpiControlInit(control);
if (returnValue != MPIMessageOK) {
      fprintf(stderr, "mpiControlInit(0x%x) returns 0x%x: %s\n",
                        control,
                        returnValue,
                        mpiMessage(returnValue, NULL));

      exit(returnValue);
}
```

2

```
/* Set up the pointers to the Xmp and Buffer data */
returnValue =
      mpiControlMemory(control,
            &Xmp,
            &external);
meiASSERT(returnValue == MPIMessageOK);
```

3

```
/* Create motor object */
motor =
      mpiMotorCreate(control,
                              motorNumber);
msgCHECK(mpiMotorValidate(motor));


/* Create axis object */
axis =
      mpiAxisCreate(control,
4                             motorNumber);
msgCHECK(mpiAxisValidate(axis));


/* Create motion object */
motion =
      mpiMotionCreate(control,
                                 motorNumber,
                                 axis);
msgCHECK(mpiMotionValidate(motion));


/* Set up for SIM4 */
returnValue =
      mpiMotorConfigGet(motor,
                                   NULL,
                                   &motorConfigXmp);
msgCHECK(returnValue);

5
motorConfigXmp.SIM4 = TRUE;

returnValue =
      mpiMotorConfigSet(motor,
                                   NULL,
                                   &motorConfigXmp);
msgCHECK(returnValue);


/* Set up Capture for SIM4 */

/* Calculate default capture number for axisNumber */
captureNumber = (motorNumber/MEIXmpMotorsPerBlock) * MEIXmpMaxLatches +
      (motorNumber % MEIXmpMotorsPerBlock) * 2;

6
/* Create capture object for captureNumber */
capture =
      mpiCaptureCreate(control,
                                  captureNumber);
msgCHECK(mpiCaptureValidate(capture));


/* Disable capture */
returnValue =
7     mpiCaptureArm(capture,
                              FALSE);
msgCHECK(returnValue);
meiPlatformSleep(100);
```

```
/* Set capture parameters for trigger on SIM4 Index pulse & ENCA & ENCB*/
returnValue =
      mpiCaptureConfigGet(capture,
                                  &captureConfig,
                                  NULL);
msgCHECK(returnValue);

captureConfig.trigger.mask =MEIMotorInputSIM4_INDEX |
                                              MEIMotorInputSIM4_ENCB |
                                              MEIMotorInputSIM4_ENCA;
captureConfig.trigger.pattern = captureActiveEdge ? (MEIMotorInputSIM4_INDEX |
                                                  MEIMotorInputSIM4_ENCB |
                                                  MEIMotorInputSIM4_ENCA) : 0;


returnValue =
      mpiCaptureConfigSet(capture,
                                  &captureConfig,
                                  NULL);
msgCHECK(returnValue);
```

8

```
/* Configure CaptureMode */
blockNumber = motorNumber / MEIXmpMotorsPerBlock;
motorInBlock = motorNumber % MEIXmpMotorsPerBlock;
returnValue =
      meiControlServiceCommand(control,
                               &external->ServiceCmdBuffer.
                               Block[blockNumber].Capture.CaptureMode[motorInBlock],
                               &CaptureMode[motorNumber],
                               1,
                               &Xmp->Block[blockNumber].ServiceCtrl,
                               MPIWaitMSEC * 1000);
msgCHECK(returnValue);
```

9

```
/* Configure Home Event action */
returnValue =
      mpiMotorEventConfigGet(motor,
                                     MPIEventTypeHOME,
                                     &eventConfig,
                                     NULL);
msgCHECK(returnValue);

eventConfig.action = MPIActionNONE;/* no event */
eventConfig.trigger.polarity = TRUE;

returnValue =
      mpiMotorEventConfigSet(motor,
                                     MPIEventTypeHOME,
                                     &eventConfig,
                                     NULL);
msgCHECK(returnValue);
```

10

```
/* Arm the capture */
returnValue =
      mpiCaptureArm(capture,
                    TRUE);
msgCHECK(returnValue);
meiPlatformSleep(100);
```

11

```
        /* Delete the handles */
        returnValue = mpiMotionDelete(motion);
        msgCHECK(returnValue);

        returnValue = mpiCaptureDelete(capture);
        msgCHECK(returnValue);

        returnValue = mpiAxisDelete(axis);
        msgCHECK(returnValue);

        returnValue = mpiMotorDelete(motor);
        msgCHECK(returnValue);

        returnValue = mpiControlDelete(control);
        msgCHECK(returnValue);

    return ((int)returnValue);
}
```