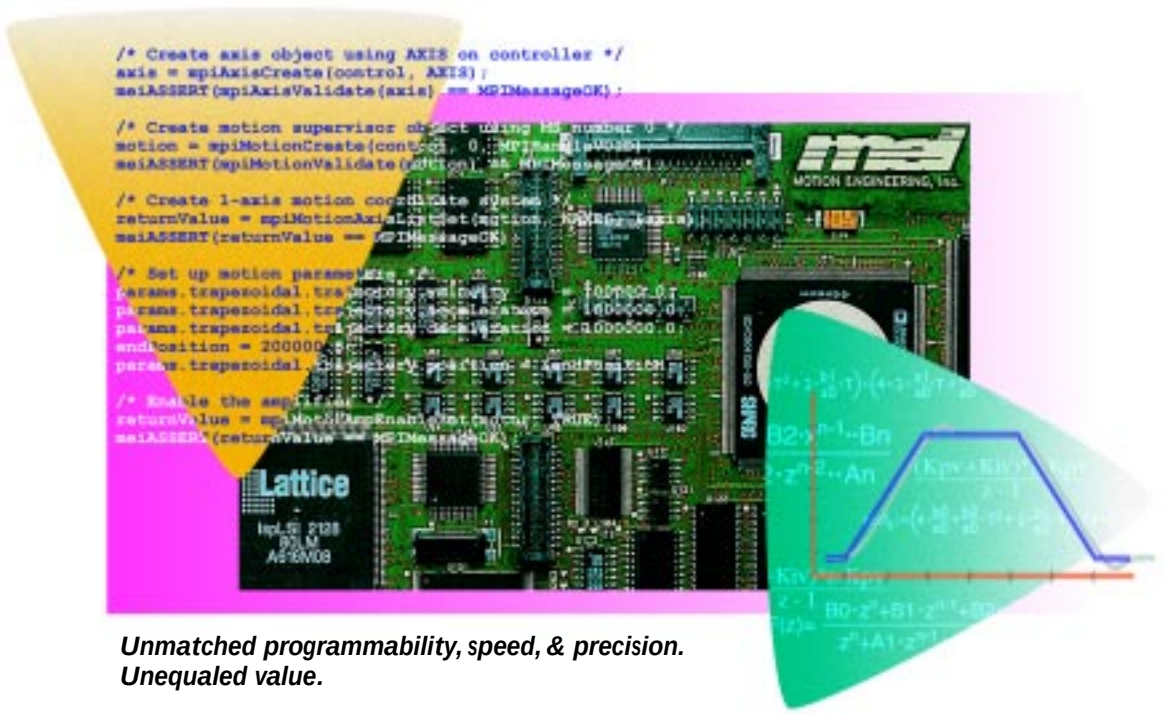
C-programmable motion control

XMP SERIES

PRODUCT OVERVIEW



Unmatched programmability, speed, & precision.
Unequaled value.

Motion Engineering, Inc.

XMP SERIES MOTION CONTROLLERS

MEI's XMP Series motion controllers offer a level of programmability, speed, and precision unmatched by any other PC-based controller. With its software-defined capabilities, the XMP can be readily customized to fit the requirements of the most demanding OEM applications.

- *C/C++ programmable*
- *Object-oriented API, Motion Programming Interface*
- *32-bit floating point DSP*
- *Servo update rates: 10 kHz for 8 axes (5 kHz for 16 axes)*
- *Up to 16 axes of servos or steppers*
- *Support for Windows NT, Windows 95/98, VenturCom, and other real-time operating systems*
- *Optional on-board sinusoidal commutation for up to 16 axes*
- *Optional scale interpolation (up to 1,024x)*

The XMP's object-oriented API, the Motion Programming Interface (MPI), lets you create complex, multi-threaded applications in C or C++. Commands and data pass from host to XMP across a high-speed PCI bus via a 32-bit direct memory interface.

A 32-bit floating-point DSP provides the XMP with the processing bandwidth to control up to 16 axes. This 150 MFLOPS DSP also delivers servo update rates of as high as 10 kHz for 8 axes (5 kHz for 16 axes).

XMP options extend performance even further. These include on-board sinusoidal commutation for up to 16 axes and scale interpolation for submicron position accuracy.

XMP-CPCI

XMP-PCI

DEVELOPMENT SOFTWARE

Object-oriented programming

The XMP is programmed using the MPI (Motion Programming Interface), an object-oriented, C/C++ programming interface. You access all XMP resources—hardware and firmware—through the MPI.

The MPI's object-oriented architecture lets you build your motion code the same way you build your machine: by creating individual software objects that reflect the components and actions of the hardware.

An MPI application creates objects for the motion controller and its resources (such as axes, input and output devices, or firmware tasks) and manipulates them using various methods.

The MPI is written entirely in ANSI-compatible C code for maximum portability. With rules and methodology consistent with other object-oriented interfaces, the MPI simplifies integration of motion code with other software components in your machine.

MPI objects

Objects in the MPI include:

Control—manages a motion controller device and handles all communication with the firmware

Axis—is associated with a single physical axis on a motion controller. When required, an Axis object can control multiple motors that perform as a single axis, such as a gantry system.

Motor—is associated with a physical motor. Motor objects handle the amplifier, I/O bits, and events associated with the motor.

Motion—maintains an ordered list of Axis objects that specify the coordinate system for all motions to be performed. Motions are defined by type (trapezoidal, S-curve, parabolic, etc.) and parameters (position, velocity, acceleration, etc.).

Position—manages the firmware resource that calculates control algorithm for an axis.

EventMgr (Event Manager)—receives asynchronous events (such as an end-of-move or a position compare) from the firmware. In response, an Event object is generated by the firmware and sent to any threads waiting for those events.

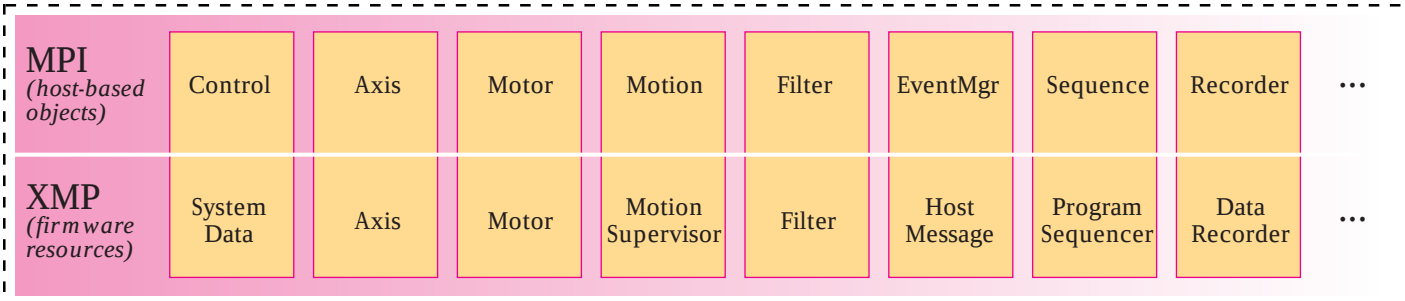
Sequence—is a linked list of motion commands executed directly on the firmware.

Recorder—collects motion information from the firmware to be accessed by external tools such as Motion Scope, Motion Console, or other graphing/analysis packages.

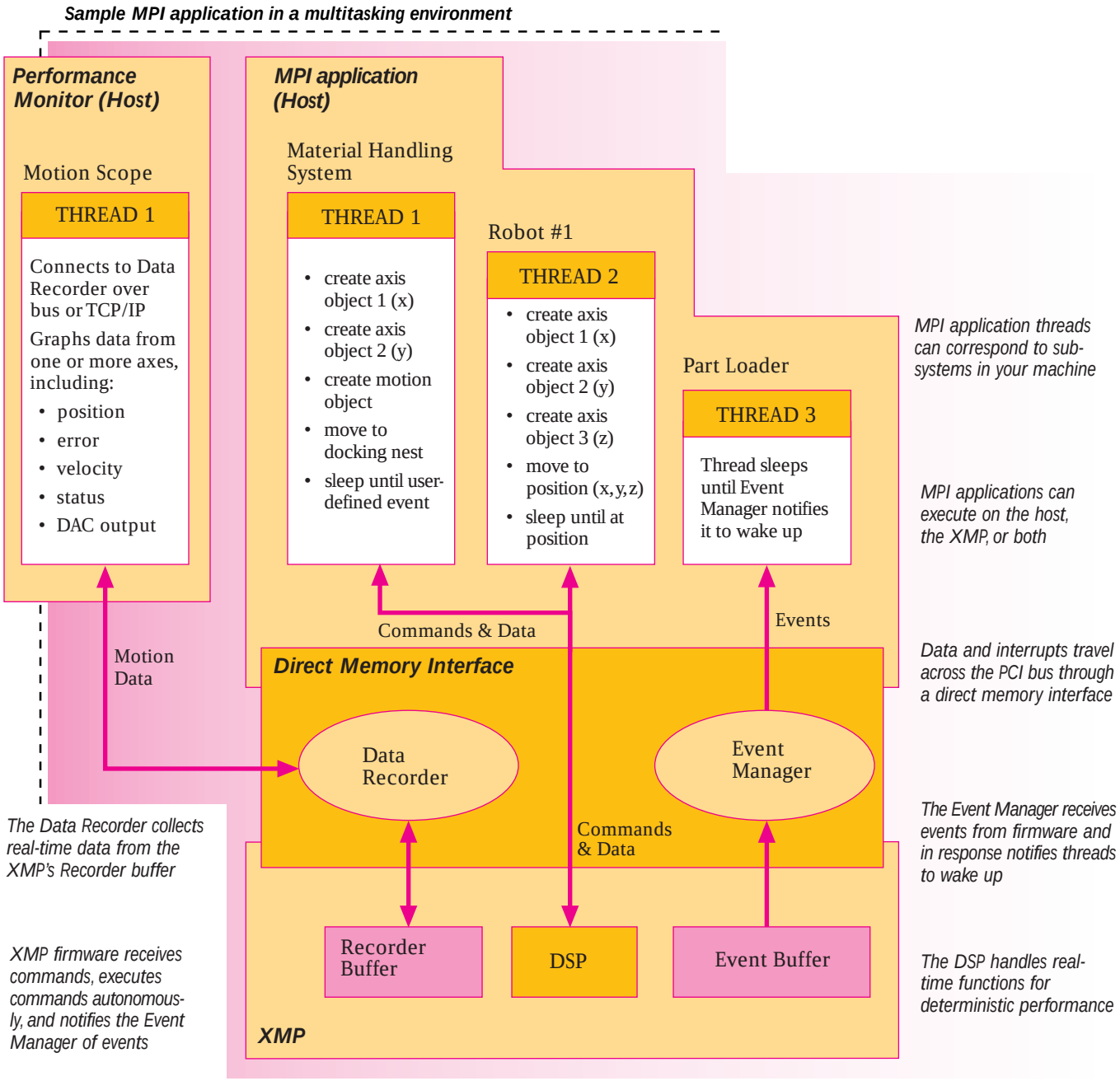
```
AxisActualPositionGet()
AxisActualPositionSet()
AxisConfigGet()
AxisConfigSet()
AxisControl()
AxisFlashConfigGet()
AxisFlashconfigSet()
AxisMemoryGet()
AxisMemorySet()
AxisNumber()
AxisStatus()
```

Selected Axis object methods

Each Motion Programming Interface (MPI) object has a corresponding firmware resource



PROGRAMMING THE XMP



Debugging tools

The MPI's trace feature displays a stream of messages to let you view an application's progress as it executes.

Trace switches can be turned on to display:

- when a library function is entered or exited
- when host memory is allocated or freed
- when XMP memory is read or written
- when an object is validated
- when a resource lock is released or obtained
- and many other conditions

Once development is complete, you can run the application with the trace feature turned off if desired.

In addition to the trace feature, the MPI provides runtime error checking to validate that applications are running correctly. For example, this feature checks whether bad data is passed to a function, pointers are correct, and parameters are valid.

When runtime checking finds an error, it can stop the application and display the source file name and line number where the error occurred.

Real-time motion control

The host CPU communicates with the XMP controller via direct memory reads and writes over a 32-bit PCI or CompactPCI bus.

Depending on application requirements, MPI programs can execute on the host, on the XMP, or be divided between both. For example, tasks that require deterministic control can be completely off-loaded to the XMP while less time-critical tasks can remain with the host.

When tasks execute on the XMP, the host downloads commands into a buffer managed by one or more Program Sequencers. Program Sequencers monitor how many commands remain in the buffer and trigger the host to download more commands when needed. This allows command sequences to be of any length.

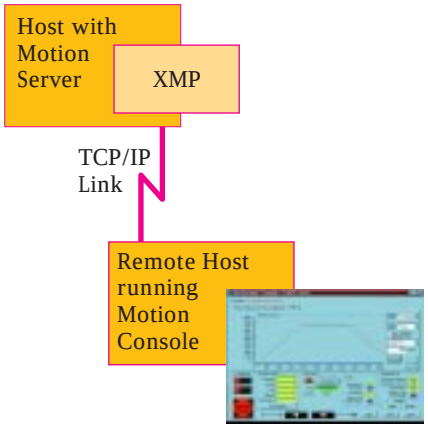
Operating system support

The MPI was designed for multi-threaded environments and supports Windows NT, Windows 95/98, VenturCom real-time extensions, and other real-time operating systems. The reentrant library includes operating system locking mechanisms to control access to shared resources.

TCP/IP remote connections

For remote motion control and diagnostics, the MPI supports a socket-based client-server mode of operation running over TCP/IP.

TCP/IP connections



Sample MPI motion routine

```
/* Create axis object using AXIS on controller */
axis = mpiAxisCreate(control, AXIS);
meiASSERT(mpiAxisValidate(axis) == MPIMessageOK);

/* Create motion supervisor object using MS number 0 */
motion = mpiMotionCreate(control, 0, MPIHandleVOID);
meiASSERT(mpiMotionValidate(motion) == MPIMessageOK);

/* Set up motion parameters */
params.trapezoidal.trajectory.velocity     = 100000.0;
params.trapezoidal.trajectory.acceleration = 1000000.0;
params.trapezoidal.trajectory.deceleration = 1000000.0;
endPosition = 200000.0;
params.trapezoidal.trajectory.position = &endPosition;

/* Enable the amplifier */
returnValue = mpiMotorAmpEnableSet(motor, TRUE);
meiASSERT(returnValue == MPIMessageOK);

/* Start motion */
returnValue = mpiMotionStart(motion,
MPIMotionTypeTRAPEZOIDAL, &params);
```

DEVELOPMENT UTILITIES

Motion Console

You set up, tune, and configure XMP controllers using Motion Console. With this Windows-based program, you can verify system wiring and spin motors with just a few mouse clicks.

Motion Console capabilities include installing and configuring multiple controllers, modifying tuning parameters, tuning the system, checking axis status and more.

Motion Console includes an oscilloscope graphing function to chart position, voltage, velocity, and error. You can graph motion in real time while tuning an axis or plot sampled data from a previous motion sequence.

For remote configuration and tuning, Motion Console can communicate with an XMP system over a TCP/IP link.

Motion Scope

You monitor the real-time performance of your MPI motion applications using Motion Scope, a powerful debugging tool for retrieving and graphing motion and I/O data. Motion Scope combines the capabilities of an oscilloscope with a logic analyzer, accessing data from the XMP in real time.

Motion Scope can display in continuous, sampled, and triggered modes. You can simultaneously view multiple graphs to compare data of different types or from different sources. Each graph can be independently scaled and configured with different display options. You can save graphs for future reference and output ASCII data for import into other analysis tools.

Like Motion Console, Motion Scope can access data from a remote XMP system via a TCP/IP communication link.

Recorder

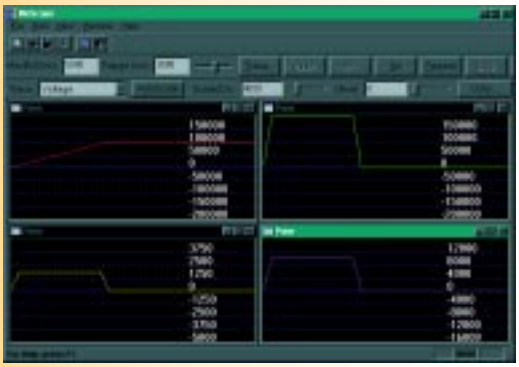
The Recorder, an object in the MPI, collects performance and position data from the XMP, including command and actual position, error, command and actual velocity, status, and DAC output.

XMP system data collected by the Recorder can be displayed using Motion Console, Motion Scope, or other graphic/analysis tools.

Motion Console and Motion Scope can access a local or remote XMP

Motion Console graphing window

Motion Console configuration window (Japanese version)



Motion Scope analysis with four panes

PID and PIV control algorithms

The XMP can provide either PID or PIV (velocity feedback) control algorithm for each axis. Both use velocity, acceleration, and friction feedforward.

With PID compensation, the single-loop system uses only position feedback. A PID compensator filters the position error signal. The derivative term of the PID compensation provides system dampening.

Under PIV compensation, an inner velocity loop provides system dampening. A velocity estimate is derived from position feedback. The velocity loop also uses a PI compensator. Additional compensation can be implemented using biquad filter blocks if required.

With its powerful processor and flexible architecture, the XMP can implement and quickly execute custom control algorithms, including state feedback and state observers.

Notch filter design toolkit

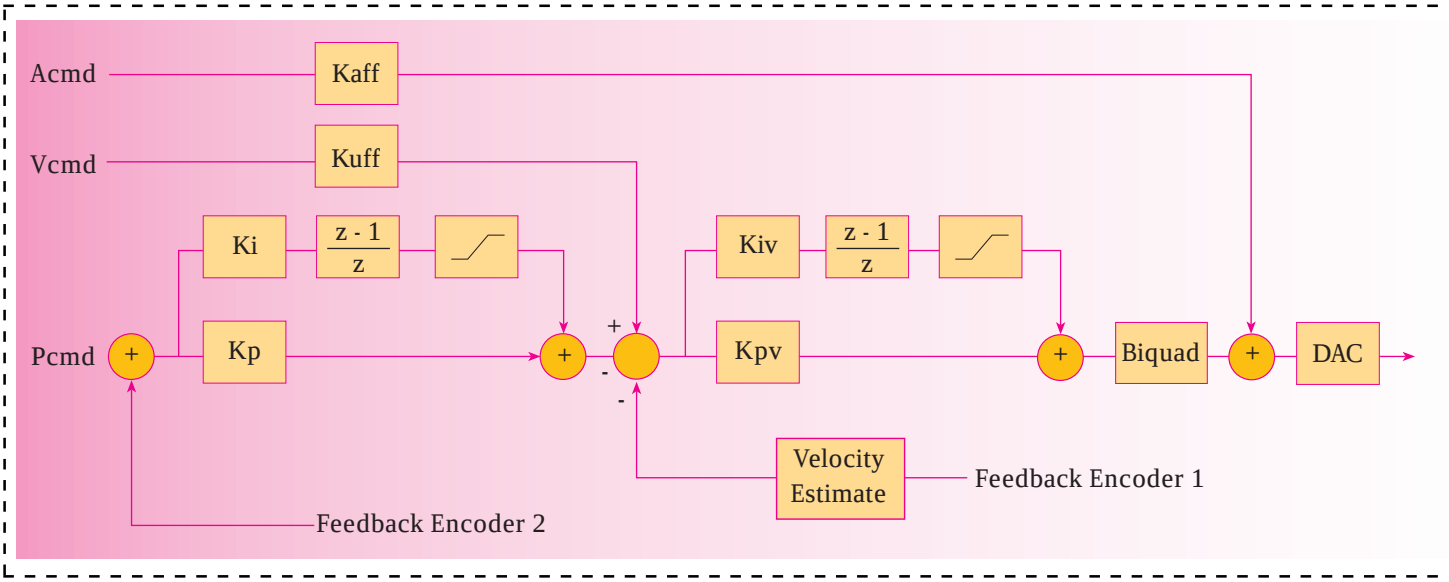
To improve move times and system stability, the XMP's easy-to-use filter design toolkit lets you implement multi-stage notch and low-pass filters.

The filter design toolkit provides a graphical interface for specifying the low-pass cutoff frequencies (or the center frequency and width for notch filters). The toolkit automatically calculates digital coefficients to be downloaded to the XMP's cascading biquad filter or saved to a file.

You can design different filters for each axis. Up to six biquad stages/axis (12th order filter) are supported.

Filter design toolkit

XMP PIV algorithm



ADVANCED CAPABILITIES

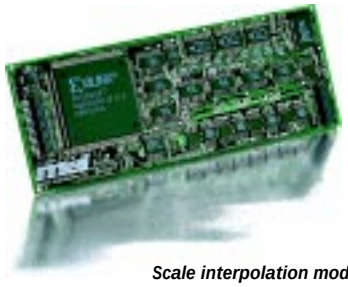
Optional scale interpolation

For submicron position accuracy, an optional scale interpolation module can increase scale resolution to 1,024x.

Each scale interpolation module supports up to four axes (with simultaneous sampling) and accepts feedback in voltage or current modes. The XMP supports up to 16 axes of interpolation.

The scale interpolation module can interpolate data from a 10 µm encoder to within 2.5 nm (12 bits of interpolation). Direct connection to a scale allows an equivalent quadrature count rate of 500 MHz. The module eliminates expensive external encoder interpolators and reduces system cost and complexity.

The scale interpolation module is ideal for high-resolution x-y positioning systems and improves the accuracy of the XMP's position capture and compare features to submicron levels.



Scale interpolation module

Position capture

The XMP can capture the exact position of one or more axes whenever a specified trigger condition occurs. This feature is useful for homing, probing, wafer centering, wafer mapping, and coordinate measurement applications.

Triggers for position capture can be a combination of the following inputs:

- home
- index
- positive or negative overtravel
- transceiver

Several axes can be captured simultaneously from one or more input triggers.

Position capture is implemented in hardware with latency under 2 µsec for up to 16 axes.

Position compare

The XMP includes comparison hardware to generate output signals based on the precise position of one or more axes. This position compare capability can be used to trigger devices such as imaging subsystems.

Ten compare registers for each group of four axes can be used singly or in combination to provide complex outputs such as:

- output ON if x and y axes are both inside a specified position window
- outputs ON and OFF at predefined points during motion
- high speed repetitive output sequence

Because output signals are hardware based, latency is under 2 µsec for up to 16 axes.

Motion functions

Multi-axis synchronized motion
Multi-axis coordinated motion
Optional scale interpolation
Optional sinusoidal commutation
Trapezoidal, parabolic, & S-curve profiles
Symmetric & asymmetric profiles
Velocity moves
Custom trajectories
On-the-fly trajectory modification
Velocity-generated events
Two-dimensional compensation tables
Post-PID cascading biquad filters
Low-pass and notch filter toolkit
Gantry algorithms
On-board settling
Electronic gearing
Electronic camming
Dual-loop support
Position compare
Position capture
Laser power control
Circular interpolation

On-the-fly trajectory modification

In the XMP, a motion trajectory can be changed at any time and as often as needed by updating acceleration, velocity, and/or position (more than one can be changed at the same time). The DSP automatically recalculates the trajectory in real time.

On-the-fly trajectory modification is useful in high-speed applications that integrate vision systems, such as electronic assembly machines that need to verify component orientation before placement.

On-board settling

The XMP performs on-board settling to more quickly and accurately determine in-position status. To do this, the XMP compares actual values with three parameters (position error tolerance, velocity error tolerance, and duration or each). Once all three criteria are met, the XMP generates a settled event.

By increasing machine throughput, on-board settling can improve performance in chip shooters and semiconductor manufacturing machines.

2D compensation tables

To compensate for surface irregularities when moving a mechanism over an x-y stage or high-precision positioning system, the XMP supports two-dimensional compensation tables. This yields higher accuracy in submicron positioning systems.

The user-supplied compensation table is downloaded into XMP memory. The XMP automatically applies this compensation information to optimize motion profiles in real time.

Custom accessory modules

XMP capabilities can be extended to meet advanced, application-specific requirements.

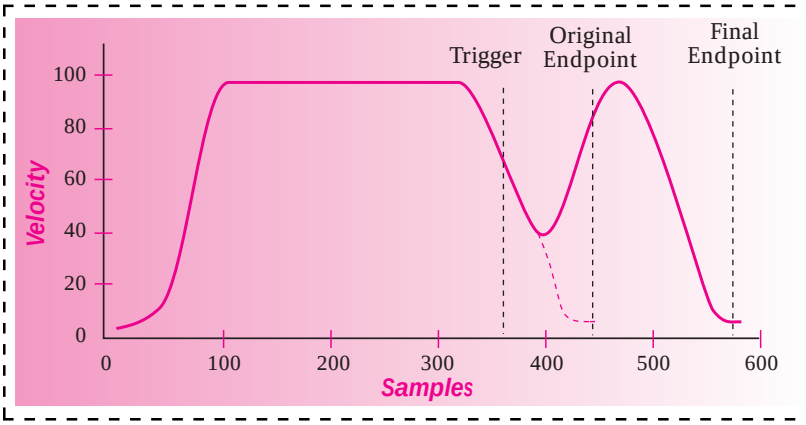
For volume OEM customers, custom mezzanine modules can be developed to customer-specific interfaces or devices such as custom I/O, specialized position encoders, system sensors, etc.

On-board sinusoidal commutation

To maintain smooth motor operation and low torque ripple—especially at low speeds—XMP controllers are available with optional dual DACs per axis to support on-board sinusoidal commutation for up to 16 axes.

Using feedback from a high-resolution encoder, the XMP precisely energizes motor windings by generating the "A" and "B" sinusoidal signals (the servo amplifier derives the third signal).

The XMP sinusoidal commutation option provides several initialization routines, including phase step, Hall sensor, and dithering. For optimal performance at high velocities, the sinusoidal commutation option also supports phase advance.



On-the-fly trajectory modification lets you change trajectory at any time and as often as needed

HARDWARE FEATURES

Powerful DSP core

The hardware architecture of the XMP is centered around the 32-bit floating-point SHARC DSP (150 MFLOPS). The SHARC gives the XMP the power to update 8 axes at speeds up to 10 kHz (16 axes at up to 5 kHz).

The SHARC DSP also provides the processing bandwidth required to handle both motion and trajectory calculations, freeing the host from real-time control functions. By automatically calculating motion algorithms in real time, the XMP can change trajectory on-the-fly.

Fast bus communications

The host CPU communicates with the XMP via direct memory reads and writes over the 32-bit PCI or Compact PCI bus. The XMP transfers data across the bus at PCI bus speeds.

Modular architecture

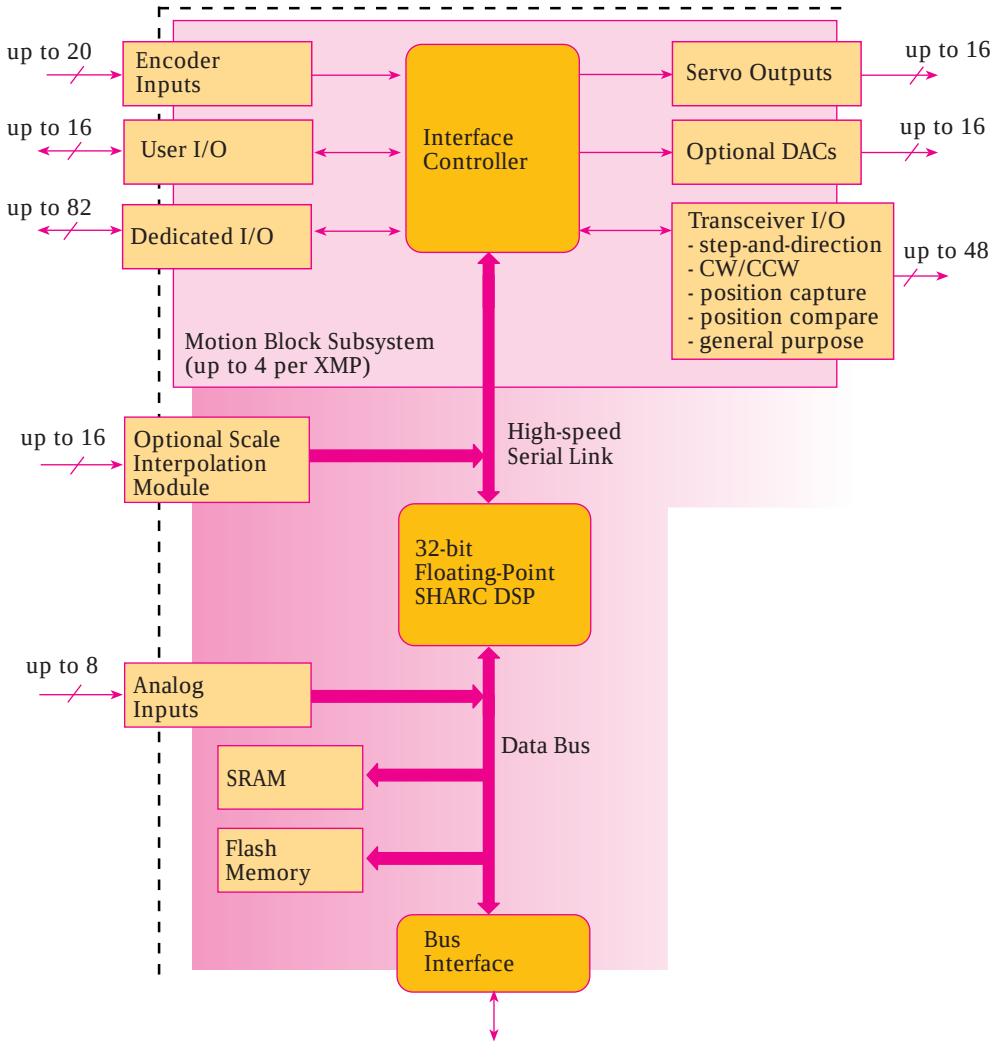
Individual axis are grouped into motion blocks, with four axes per block. Up to two motion blocks reside on the XMP main board. In systems with 12 or 16 axes, additional motion blocks reside on an expansion board, connected to the main board via a 40 Mbit/sec serial link.

Each motion block includes five encoder inputs as well as an interface controller that provides high-speed communications between the DSP and all motion I/O functions.

Position feedback capabilities

The XMP supports position feedback from incremental encoders (at count rates up to 40 MHz) and analog scales. The XMP also features eight true differential 16-bit analog inputs that can be used to read analog input signals or can be configured for position feedback, force feedback, and jogging.

XMP Series hardware architecture



Encoder integrity checking

MEI's Encoder integrity checking (EIC) feature monitors encoder lines and immediately shuts down a potential runaway condition caused by broken or shorted encoder wires.

EIC prevents runaway conditions to safeguard against hardware malfunctions and wiring errors.

I/O capabilities

The XMP provides three types of I/O as follows:

Dedicated—up to 82 lines, opto-isolated. Five lines per axis: home, positive limit, negative limit, and amp fault inputs plus amp enable output. Also master E-Stop and reset inputs per controller.

Transceiver—up to 48 lines, fast EIA-422. Used to implement stepper outputs or for position capture inputs and position compare outputs.

User—up to 16 lines, opto-isolated.

Watchdog timeout

The XMP's watchdog timeout is a fail-safe hardware feature that provides for an orderly shutdown of motion events. If the fault monitor circuitry detects a firmware malfunction, the watchdog timer can trigger a software reset or other user-specified event.

High-density connectors

The XMP connects to external devices through high-density, shielded connectors that comply with the SCSI-4 VHDCI standard. XMP connectors offer a low profile that enables up to 272 I/O connections to be brought out in a single PCI slot. Shielded cable assemblies are also available from MEI; one is required for every two axes.

Hardware features

- *up to 16 axes servos and steppers*
- *32-bit direct memory interface*
- *16-bit servo output resolution*
- *4 MHz digitally synthesized step output*
- *shielded high-density connectors*
- *16-bit differential analog inputs*
- *analog (joystick) jogging*
- *up to 82 lines dedicated I/O (opto-isolated)*

You access XMP hardware capabilities through the Motion Programming Interface

PRODUCT EVOLUTION

Custom XMP models

MEI has an extensive base of motion control technology that can be combined with the XMP's powerful software and hardware to meet specific customer requirements.

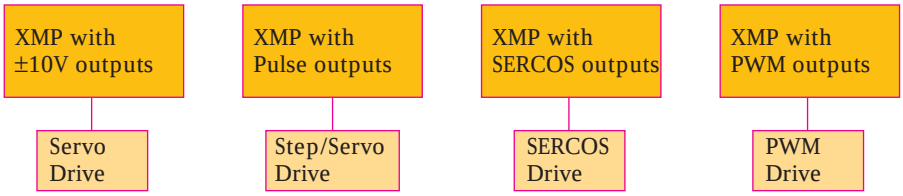
Other outputs

Along with standard ± 10V analog and stepper outputs, XMP will offer an interface to SERCOS fiber-optic devices and also support PWM outputs in the future.

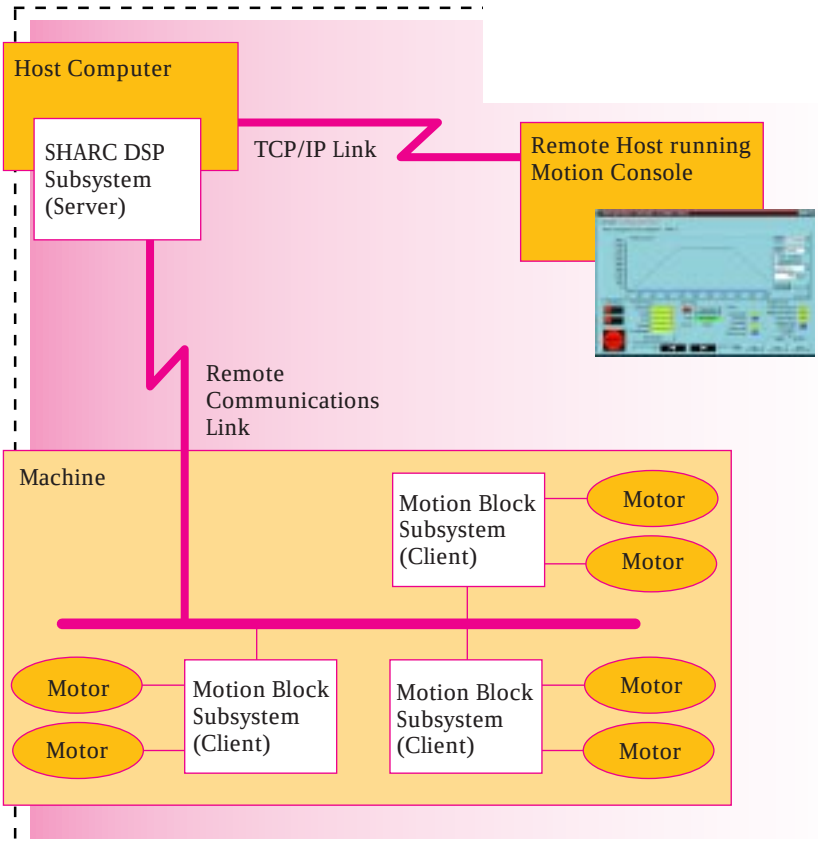
Distributed control

The XMP's modular hardware design could also be segmented to accommodate distributed control.The SHARC DSP subsystem would remain with the host as a server for individual client nodes located near the motors and drives.The DSP server would calculate motion trajectories and pass data to motion block clients across a high-speed TCP/IP or other communication network.

Current and future XMP output options



Projected XMP distributed control scenario



PINOUTS

Axes 0-1

Pin	*Signal*	*Signal*	*Pin*
1	Analog_IN_0+	Analog_IN_0-	35
2	Analog_IN_1+	Analog_IN_1-	36
3	Gnd	A_Gnd	37
4	Enc0_A+	Enc0_A-	38
5	Enc0_B+	Enc0_B-	39
6	Enc0_I+	Enc0_I-	40
7	Home0_IN	5V_OUT	41
8	Pos_Lim0_IN	Gnd	42
9	Neg_Lim0_IN	HomeLim0_Rtn	43
10	Cmd_Dac_OUT_0	A_Gnd	44
11	Aux_Dac_OUT_0	A_Gnd	45
12	Amp_Flt0_IN	Amp_Flt0_Rtn	46
13	Amp_En0_Collector	Amp_En0_Emitter	47
14	UserIO_A0	UserIO_A0_Rtn	48
15	Xcvr0A+	Xcvr0A-	49
16	Xcvr0B+	Xcvr0B-	50
17	Xcvr0C+	Xcvr0C-	51
18	Enc1_A+	Enc1_A-	52
19	Enc1_B+	Enc1_B-	53
20	Enc1_I+	Enc1_I-	54
21	Home1_IN	5V_OUT	55
22	Pos_Lim1_IN	Gnd	56
23	Neg_Lim1_IN	HomeLim1_Rtn	57
24	Cmd_Dac_OUT_1	A_Gnd	58
25	Aux_Dac_OUT_1	A_Gnd	59
26	Amp_Flt1_IN	Amp_Flt1_Rtn	60
27	Amp_En1_Collector	Amp_En1_Emitter	61
28	Gnd	Gnd	62
29	Xcvr1A+	Xcvr1A-	63
30	Xcvr1B+	Xcvr1B-	64
31	Xcvr1C+	Xcvr1C-	65
32	UserIO_A1	UserIO_A1_Rtn	66
33	RESET_IN	UserIO_A2	67
34	ESTOP_IN	UserIO_A2_Rtn	68

Pinout notes:

Pinouts shown are for axes 0-7 on XMP main board (all models).

Full interface documentation is available in XMP Motion Controller Hardware Reference.

Axes 2-3

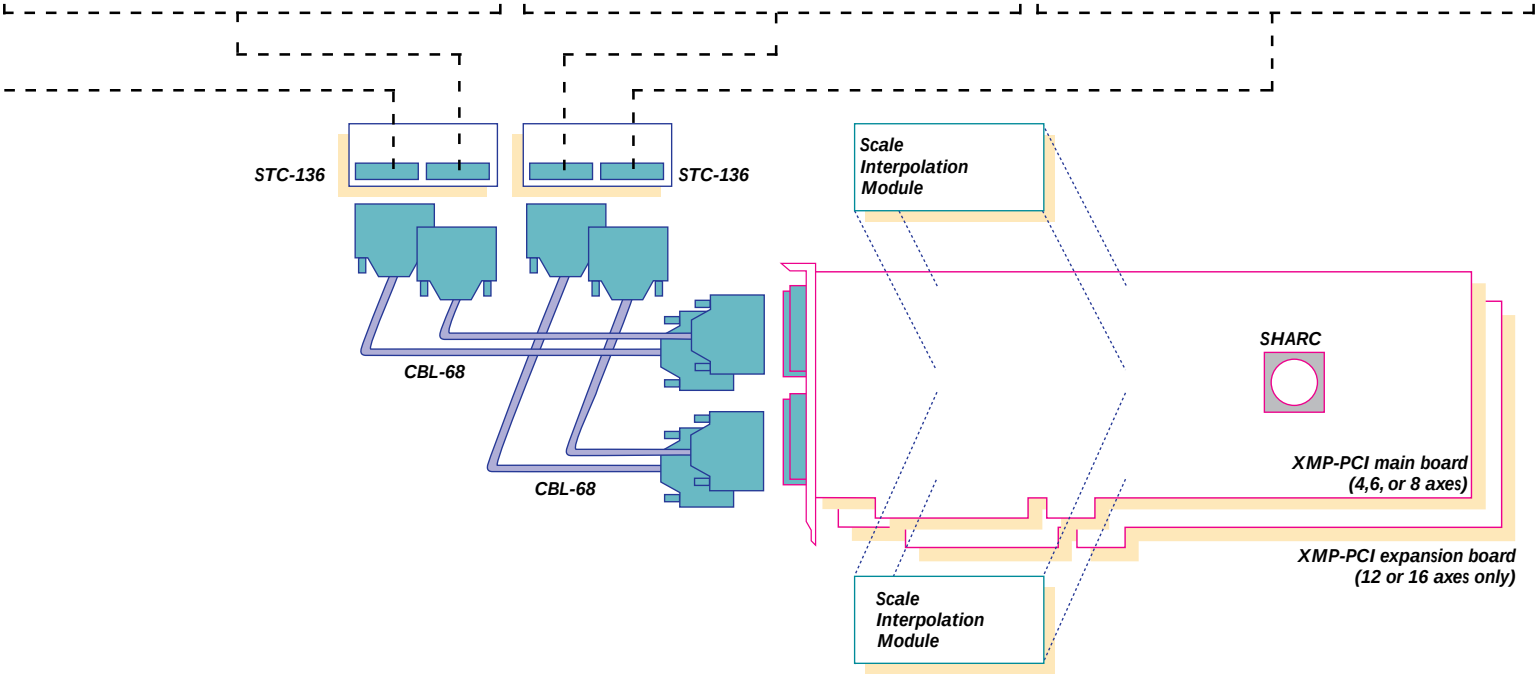
Pin	Signal	Signal	Pin
1	EncA_A+	EncA_A-	35
2	EncA_B+	EncA_B-	36
3	EncA_I+	EncA_I-	37
4	Enc2_A+	Enc2_A-	38
5	Enc2_B+	Enc2_B-	39
6	Enc2_I+	Enc2_I-	40
7	Home2_IN	5V_OUT	41
8	Pos_Lim2_IN	Gnd	42
9	Neg_Lim2_IN	HomeLim2_Rtn	43
10	Cmd_Dac_OUT_2	AGnd	44
11	Aux_Dac_OUT_2	AGnd	45
12	Amp_Flt2_IN	Amp_Flt2_IN	46
13	Amp_En2_Collector	Amp_En2_emitter	47
14	UserIO_A3	UserIO_A3_Rtn	48
15	Xcvr2A+	Xcvr2A-	49
16	Xcvr2B+	Xcvr2B-	50
17	Xcvr2C+	Xcvr2C-	51
18	Enc3_A+	Enc3_A-	52
19	Enc3_B+	Enc3_B-	53
20	Enc3_I+	Enc3_I-	54
21	Home3_IN	5V_OUT	55
22	Pos_Lim3_IN	Gnd	56
23	Neg_Lim3_IN	HomeLim3_Rtn	57
24	Cmd_Dac_OUT_3	AGnd	58
25	Aux_Dac_OUT_3	AGnd	59
26	Amp_Flt3_IN	Amp_Flt3_Rtn	60
27	Amp_En3_Collector	Amp_En3_Emitter	61
28	Gnd	Gnd	62
29	Xcvr3A+	Xcvr3A-	63
30	Xcvr3B+	Xcvr3B-	64
31	Xcvr3C+	Xcvr3C-	65
32	Gnd	AGnd	66
33	Analog_IN_2+	Analog_IN_2-	67
34	Analog_IN_3+	Analog_IN_3-	68

Axes 4-5

Pin	Signal	Signal	Pin
1	UserIO_B0	UserIO_B0_Rtn	35
2	UserIO_B1	UserIO_B1_Rtn	36
3	Gnd	Gnd	37
4	Enc4_A+	Enc4_A-	38
5	Enc4_B+	Enc4_B-	39
6	Enc4_I+	Enc4_I-	40
7	Home4_IN	5V_OUT	41
8	Pos_Lim4_IN	Gnd	42
9	Neg_Lim4_IN	HomeLim4_Rtn	43
10	Cmd_Dac_OUT_4	AGnd	44
11	Aux_Dac_OUT_4	AGnd	45
12	Amp_Flt4_IN	Amp_Flt4_IN	46
13	Amp_En4_Collector	Amp_En4_Emitter	47
14	UserIO_B2	UserIO_B2_Rtn	48
15	Xcvr4A+	Xcvr4A-	49
16	Xcvr4B+	Xcvr4B-	50
17	Xcvr4C+	Xcvr4C-	51
18	Enc5_A+	Enc5_A-	52
19	Enc5_B+	Enc5_B-	53
20	Enc5_I+	Enc5_I-	54
21	Home5_IN	5V_OUT_2	55
22	Pos_Lim5_IN	Gnd	56
23	Neg_Lim5_IN	HomeLim5_Rtn	57
24	Cmd_Dac_OUT_5+	AGnd	58
25	Aux_Dac_OUT_5+	AGnd	59
26	Amp_Flt5_IN	Amp_Flt5_Rtn	60
27	Amp_En5_Collector	Amp_En5_Emitter	61
28	Gnd	Gnd	62
29	Xcvr5A+	Xcvr5A-	63
30	Xcvr5B+	Xcvr5B-	64
31	Xcvr5C+	Xcvr5C-	65
32	Gnd	AGnd	66
33	Analog_IN_4+	Analog_IN_4-	67
34	Analog_IN_5+	Analog_IN_5-	68

Axes 6-7

Pin	Signal	Signal	Pin
1	Analog_IN_6+	Analog_IN_6-	35
2	Analog_IN_7+	Analog_IN_7-	36
3	Gnd	AGnd	37
4	Enc6_A+	Enc6_A-	38
5	Enc6_B+	Enc6_B-	39
6	Enc6_I+	Enc6_I-	40
7	Home6_IN	5V_OUT	41
8	Pos_Lim6_IN	Gnd	42
9	Neg_Lim6_IN	HomeLim6_Rtn	43
10	Cmd_Dac_OUT_6	AGnd	44
11	Aux_Dac_OUT_6	AGnd	45
12	Amp_Flt6_IN	Amp_Flt6_Rtn	46
13	Amp_En6_Collector	Amp_En6_Emitter	47
14	UserIO_B3	UserIO_B3_Rtn	48
15	Xcvr6A+	Xcvr6A-	49
16	Xcvr6B+	Xcvr6B-	50
17	Xcvr6C+	Xcvr6C-	51
18	Enc7_A+	Enc7_A-	52
19	Enc7_B+	Enc7_B-	53
20	Enc7_I+	Enc7_I-	54
21	Home7_IN	5V_OUT	55
22	Pos_Lim7_IN	Gnd	56
23	Neg_Lim7_IN	HomeLim7_Rtn	57
24	Cmd_Dac_OUT_7	AGnd	58
25	Aux_Dac_OUT_7	AGnd	59
26	Amp_Flt7_IN	Amp_Flt7_Rtn	60
27	Amp_En7_Collector	Amp_En7_Emitter	61
28	Gnd	Gnd	62
29	Xcvr7A+	Xcvr7A-	63
30	Xcvr7B+	Xcvr7B-	64
31	Xcvr7C+	Xcvr7C-	65
32	EncB_A+	EncB_A-	66
33	EncB_B+	EncB_B-	67
34	EncB_I+	EncB_I-	68



SELECTED FUNCTIONS/METHODS

Control

`ControlAddress()`	Get original address of control object
`ControlError()`	Get platform-specific error info
`ControlInit()`	Initialize Control object
`ControlInteruptEnable()`	Enable XMP interrupts
`ControlInterruptWait()`	Wait for controller interrupt
`ControlInterruptWake()`	Wake all threads waiting for controller interrupt
`ControlConfigGet/Set()`	Get/set config of Control object
`ControlType()`	Get type of Control object

Axis

`AxisActualPositionGetSet()`	Get/set actual position
`AxisCommandPositionGet/Set()`	Get/set command position
`AxisControl()`	Return handle of Control associated with an Axis
`AxisMemory()`	Set axis memory address
`AxisMemoryGet/Set()`	Get/set bytes of Axis memory and put into application memory
`AxisStatus()`	Get Axis status
`AxisTrajectoryGet/Set()`	Get Axis trajectory

Motion

`MotionAction()`	Perform specified action on motion
`MotionModify()`	Modify parameters of Motion while it is executing
`MotionNumber()`	Get index of Motion
`MotionParamsGet/Set()`	Get/set Motion parameters
`MotionPositionGet/Set()`	Get/set position parameters of all Axes associated with Motion
`MotionStart()`	Start Motion (idle state to moving state)
`MotionTrajectoryGet/Set()`	Get/set trajectories for all axes associated with Motion

Filter

`FilterConfigGet/Set()`	Get/set Position config
`FilterFeedbackGet()`	Get feedback position
`FilterHomeLatchClear()`	Clear home latch associated with Position
`FilterPositionGet/Set()`	Get/set actual and command positions
`FilterStatus()`	Get Position status
`FilterAxisAppend()`	Append axis to list of Axes
`FilterAxisCount()`	Return the number of Axes in list
`FilterAxisRemove()`	Remove Axis from list

Motor

`MotorAmpEnableGet/Set()`	Get/set state of amp enable output
`MotorAxisMapGet()`	Get object map of axes
`MotorConfigGet/Set()`	Get/set Motor configuration
`MotorEventConfigSet()`	Set Motor's event configuration
`MotorMemory()`	Get address of Motor memory
`MotorStatus()`	Get Motor status

EventMgr

`EventMgrEvent()`	Request event notification for all Notify objects on EventMgr's list
`EventMgrFlush()`	Flush pending EventMgr events
`EventMgrService()`	Get list of all pending asynchronous events
`EventMgrControlListSet()`	Get list of Control objects associated with EventMgr
`EventMgrControlListSet()`	Create a list of Control objects associated with EventMgr
`EventMgrNotifyListInsert()`	Place a Notify object after another Notify object in list
`EventMgrNotifyListGet/Set()`	Get/set list of Notify objects

Sequence

`SequenceEventNotifySet()`	Enable host notification of Sequence events
`SequenceLoad()`	Load Sequence commands into firmware
`SequenceMemory()`	Set address used to access Sequence memory
`SequenceNext()`	Get handle to next command in list
`SequencePageSize()`	Set amount of memory available for commands used by sequence
`SequenceResume()`	Resume execution of Sequence
`SequenceStart()`	Start execution of Sequence
`SequenceStep()`	Execute specified steps of a stopped sequence
`SequenceCommandInsert()`	Insert command into Sequence

Recorder

`RecorderEventNotifyGet/Set`	Get/set mask of events for which host notification has been requested
`RecorderEventReset`	Reset the events specified in event mask that are generated by Recorder
`RecorderMemoryGet`	Copy data from Recorder memory to application memory
`RecorderMemorySet`	Copy data from application memory to Recorder memory
`RecorderRecordGet`	Get records from Recorder
`RecorderStart`	Start recording data records using Recorder

XMP SPECIFICATIONS

Processor
- Analog Devices SHARC DSP
- 32-bit floating-point
- 150 MFLOPS

System interfaces
- PCI and CompactPCI busses
- 32-bit direct memory interface
- High-speed binary communications across the bus
- Contact MEI for other bus and networking options

Software development
- C/C++ programmable
- Object-oriented API: Motion Programming Interface (MPI)
- Operating systems: Windows NT, Windows 95/98, VenturCom, and selected real-time operating systems

Motion control capabilities
- Supports up to 16 axes
- Point-to-point motion
- Multi-axis coordinated motion
- Multi-axis synchronized motion
- Electronic gearing & camming
- Optional sinusoidal commutation
- Optional scale interpolation
- Trapezoidal, parabolic, and S-curve profiles
- Custom trajectories
- Asymmetric & symmetric profiles
- Velocity moves
- On-the-fly trajectory modification
- On-board settling
- 2D compensation tables
- Position capture
- Position compare
- Dual-loop support
- Velocity-generated events
- Gantry algorithms
- Circular interpolation

Kinematic ranges
- Position: 32-bit floating-point (±2.15 billion counts)
- Velocity, acceleration, and jerk: 32-bit floating-point

Servo output
- ±10V at 16-bit resolution with 16-bit monotonicity
- Pulse train output (to 4 MHz for pulse-controlled servos)
- Simultaneous update of all axes
- Outputs with high drive capability (2 kΩ load in parallel with 200 pF)
- Optional sinusoidal commutation for up to 16 axes

Servo loop update rate
- User-programmable rate
- Maximum: 10 kHz (8 axes), 5 kHz (16 axes)

Step output
- Maximum step frequency: 4 MHz
- Step/direction or CW/CCW
- Open or closed loop control
- Minimum pulse width: 200 nsec
- EIA-422 Line Driver output

SERCOS interface
- Support for up to 24 axes
- 2, 4, 8, or 16 Mbits/sec
- Digital fiber-optic connection to drives

Control algorithms
- PID or PIV control with velocity, friction, and acceleration feedforward
- Support for custom control algorithms

Filter toolkit
- Used for designing multi-stage low-pass and notch filters
- Automatically calculates digital coefficients
- Post-PID cascading biquad filter

Position feedback
- Incremental encoder: 40 MHz (10 MHz quadrature input), single-ended or differential
- EIA-422 line receivers
- Digital noise filtering
- Position capture and position compare
- Optional scale interpolation (1,024x)

Analog inputs
- 8 channels differential input
- 16-bit resolution
- Programmable input range: ±1.25V to ±10V
- Single channel bandwidth of 50 kHz
- Analog (joystick) jogging
- Force feedback

Dedicated I/O
- Five opto-isolated signals per axis
- Inputs: home, positive limit, negative limit, amp-fault
- Output: amp enable
- Dedicated system inputs for E-stop and reset
- 5-24V logic

Transceiver I/O
- Up to 48 lines
- EIA-422
- Used for step-and-direction, CW/CCW, position capture, position compare, or general purpose

User I/O
- Up to 16 lines
- Opto-isolated
- 5V or 24V
- 30 mA source or sink

Connectors
- 68-pin VHDCI connectors (SCSI-4)
- Shielded twisted-pair cables

System safety
- Encoder integrity checking prevents runaway conditions caused by broken or shorted encoder wires
- On-board watchdog timer with host handshaking
- Switched analog outputs protect on:
  - power failure/brownout
  - power-up and following reset
  - dedicated E-stop input

Environmental conditions
- Operating temperature: 0-50 degrees C
- Humidity: 20 - 90% RH, non-condensing

Motion Engineering, Inc.

Corporate Headquarters
33 South La Patera Lane
Santa Barbara, California 93117
ph 805-681-3300
fax 805-681-3311
e-mail info@motioneng.com
www.motioneng.com

Eastern Regional Office
30 Nagog Park
Boston, Massachusetts
ph 978-264-0051
fax 978-264-0057

Midwestern Technical Support Office
5519 N. Cumberland Avenue
Suite 1011
Chicago, Illinois
ph 773-631-4992
fax 773-631-4936

Philadelphia Development Office
790 Pennlyn Blue Bell Pike
Suite 204
Philadelphia, Pennsylvania
ph 215-793-4220
fax 215-793-4223

Tokyo Regional Office
Asahiko Building 4F
3-1 Kagurazaka, Shinjuku-ku
Tokyo 162 Japan
ph 03-5229-7007
fax 03-3235-5655
e-mail info@motioneng.co.jp

Nagoya Technical Support Office
102 Top Hill 2
20-2 Sokuten, Akebono-cho
Toyohashi-shi
Aichi, Japan 441
ph 0532-45-3511
fax 0532-45-5415

Rev 10/98