Motor Objects

Introduction

A **Motor** object manages a single motor on a controller. It represents the physical connections between the motor, drive, and associated I/O. The Motor object contains encoder data, limit switch, home sensor, amp fault and amp enable states, DAC outputs, and other status information.

For simple systems, there is a one-to-one relationship between the Axis, Filter and Motor objects.

| Error Messages |

Methods

Create, Delete, Validate Methods

mpiMotor**Create**	Create Motor object
mpiMotor**Delete**	Delete Motor object
mpiMotor**Validate**	Validate Motor object

Configuration and Information Methods

mpiMotor**AmpEnableGet**	Get state of amp enable output
mpiMotor**AmpEnableSet**	Set state of amp enable output
meiMotor**AmpFault**	Writes the amp fault buffer data of a motor.
meiMotor**AmpFaultClear**	Clears the amp fault buffer data of a motor.
meiMotor**AmpWarning**	Writes the amp warning buffer of the motor.
meiMotor**AmpWarningClear**	Clears the motor's amp wanring message buffer.
mpiMotor**AxisMapGet**	Get object map of axes
meiMotor**CommutationModeGet**	Gets the commutation mode of a motor.
meiMotor**CommutationModeSet**	Sets the commutation mode of a motor.
mpiMotor**ConfigGet**	Get motor configuration
mpiMotor**ConfigSet**	Set motor configuration
meiMotor**ConfigStepper**	Configures a motor for stepper mode.
meiMotor**DacConfigGet**	Get a Motor's (motor) Dac configuration
meiMotor**DacConfigSet**	Set a Motor's (motor) Dac configuration
meiMotor**DacFlashConfigGet**	
meiMotor**DacFlashConfigSet**	
mpiMotor**DedicatedIn**	
mpiMotor**DedicatedOutGet**	
mpiMotor**Feedback**	Get feedback position
mpiMotor**FlashConfigGet**	Get flash config of motor

mpiMotor**FlashConfigSet**	Set flash config of motor
mpiMotor**GeneralIn**	
mpiMotor**GeneralOutGet**	
mpiMotor**GeneralOutSet**	
meiMotor**Info**	Get information about the network, node, and drive interface
meiMotor**PhaseFindStatus**	
mpiMotor**Status** / meiMotor**Status**	Get motor status

Event Methods

mpiMotor**EventConfigGet**	Get motor's event configuration
mpiMotor**EventConfigSet**	Set motor's event configuration
mpiMotor**EventNotifyGet**	Get motor's event mask for host notification.
mpiMotor**EventNotifySet**	Set motor's event mask for host notification.
mpiMotor**EventReset**	Reset events specified in event mask

Memory Methods

mpiMotor**Memory**	Get address of motor memory
mpiMotor**MemoryGet**	Copy motor memory to application memory
mpiMotor**MemorySet**	Copy application memory to motor memory

Action Methods

meiMotor**EncoderReset**	Clears encoder faults.
meiMotor**MultiTurnReset**	Clears the SynqNet drive multi-turn data for absolute type encoders.
meiMotor**PhaseFindAbort**	
meiMotor**PhaseFindStart**	

Relational Methods

mpiMotor**Control**	Get handle to associated Control object
mpiMotor**FilterMapGet**	Get object map of associated Filters
mpiMotor**FilterMapSet**	Set the Filters using object map
mpiMotor**Number**	Get index number of motor (in Control list)

Other Methods

meiMotor**EncoderRatio**	Get encoder ratio from the XMP.

Data Types

MEIMotor**AmpFaults**

MEIMotor**AmpFaultMsg**

MEIMotor**AmpWarnings**

MEIMotor**AmpWarningMsg**

MPIMotor**Brake**

MPIMotor**BrakeMode**

MPIMotor**Config** / MEIMotor**Config**

MEIMotor**DacConfig**

MEIMotor**DacChannelConfig**

MEIMotor**DacChannelStatus**

MEIMotor**DacStatus**

MPIMotor**DedicatedIn**

MPIMotor**DedicatedOut**

MEIMotor**DemandMode**

MEIMotor**DisableAction**

MPIMotor**Encoder** / MEIMotor**Encoder**

MPIMotor**EncoderFault**

MPIMotor**EncoderFaultMask**

MEIMotor**EncoderModulo**

MEIMotor**EncoderRatio**

MEIMotor**EncoderReverseModulo**

MEIMotor**EncoderSsiConfig**

MEIMotor**EncoderType**

MPIMotor**EventConfig** / MEIMotor**EventConfig**

MPIMotor**EventTrigger**

MEIMotor**FaultBit**

MEIMotor**FaultConfig**

MEIMotor**FaultMask**

MPIMotor**Feedback**

MPIMotor**GeneralIo**

MEIMotor**Info**

MEIMotor**InfoDedicatedIn**

MEIMotor**InfoDedicatedOut**

MEIMotor**InfoGeneralIo**

MEIMotor**InfoNodeType**

MEIMotor**IoConfig**

MEIMotor**IoConfigIndex**

MEIMotor**IoType**

MEIMotor**IoTypeMask**
MPIMotor**Message** / MEIMotor**Message**
MEIMotor**PhaseFindDriveMsg**
MEIMotor**PhaseFindState**
MEIMotor**PhaseFindStatus**
MEIMotor**SsiInput**
MEIMotor**Status**
MEIMotor**StatusOutput**
MEIMotor**Stepper**
MEIMotor**StepperPulse**
MEIMotor**StepperPulseType**
MEIMotor**StepperStatus**
MPIMotor**Type**

Macros

mpiMotor**EncoderFaultMaskBIT**

Constants

MEIMotor**AmpFaultsMAX**
MEIMotor**AmpMsgMAX**
MEIMotor**AmpWarningsMAX**

mpiMotorCreate

Declaration

```
const MPIMotor mpiMotorCreate(MPIControl control,
                              long       number);
```

Required Header: stdmpi.h

Description

mpiMotorCreate creates a Motor object associated with the motor identified by ***number***, and located on the motion controller (***control***). MotorCreate is the equivalent of a C++ constructor.

Return Values	
handle	to a Motor object.
MPIHandleVOID	if the Motor object could not be created.

See Also

mpiMotorDelete | mpiMotorValidate

mpiMotorDelete

Declaration

```
long mpiMotorDelete(MPIMotor motor)
```

Required Header: stdmpi.h

Description

mpiMotorDelete deletes a Motor object and invalidates its handle (***motor***). *MotorDelete* is the equivalent of a C++ destructor.

Return Values	
MPIMessageOK	

See Also

mpiMotorCreate | mpiMotorValidate

mpiMotorValidate

Declaration

```
long mpiMotorValidate(MPIMotor motor)
```

Required Header: stdmpi.h

Description

mpiMotorValidate validates a Motor object and its handle (***motor***).

motor	a handle to the Motor object

Return Values	
MPIMessageOK	

See Also

mpiMotorCreate | mpiMotorDelete

mpiMotorAmpEnableGet

Declaration

```
long mpiMotorAmpEnableGet(MPIMotor  motor,
                          MPI_BOOL*ampEnable)
```

Required Header: stdmpi.h
Change History: Modified in the 03.03.00

Description

mpiMotorAmpEnableGet gets the state of the amp enable output for a Motor (***motor***) and writes it in the location pointed to by ***ampEnable***. Note that the actual state of amp enable output also depends upon the actual wiring and the polarity chosen in the instance of the MPIMotorConfig structure.

If "ampEnable" is	*Then*
FALSE (0)	the amp is disabled
TRUE (1)	the amp is enabled

Return Values	
MPIMessageOK	

See Also

MPIMotorConfig | mpiMotorAmpEnableSet

mpiMotorAmpEnableSet

Declaration

```
long mpiMotorAmpEnableSet(MPIMotor     motor,
                          MPI_BOOL     ampEnable)
```

Required Header: stdmpi.h
Change History: Modified in the 03.03.00

Description

mpiMotorAmpEnableSet sets the state of the amp enable output for a Motor (***motor***) to ***ampEnable***. Note that the actual state of amp enable output also depends upon the actual wiring and the polarity chosen in the instance of the MPIMotorConfig structure.

If "ampEnable" is	*Then*
FALSE (0)	the amp will be disabled
TRUE (1)	the amp will be enabled

Return Values	
MPIMessageOK	

See Also

MPIMotorConfig | mpiMotorAmpEnableGet

meiMotorAmpFault

Declaration

```
long meiMotorAmpFault(MPIMotor           motor,
                      MEIMotorAmpFaults  *fault);
```

Required Header: stdmei.h

Description

meiMotorAmpFault reads the amp fault buffer from a motor and writes the data into a structure pointed to by fault.

motor	a handle to the Motor object
***fault**	a pointer to a structure containing the number of amp faults, their coded values and message strings. See MEIMotorAmpFaults.

Return Values	
MPIMessageOK	

See Also

meiMotorAmpFaultClear | meiMotorAmpWarning | meiMotorWarningClear

meiMotorAmpFaultClear

Declaration

```
long meiMotorAmpFaultClear(MPIMotor    motor);
```

Required Header: stdmei.h

Description

meiMotorAmpFaultClear flushes the motor's amp fault message buffer. The number of amp faults is set to zero, the coded values are set to zero, and the messages are cleared.

motor	a handle to the Motor object

Return Values	
MPIMessageOK	

See Also

meiMotorAmpFault | meiMotorAmpWarning | meiMotorWarningClear

meiMotorAmpWarning

Declaration

```
long meiMotorAmpWarning(MPIMotor              motor,
                        MEIMotorAmpWarnings   *warning);
```

Required Header: stdmei.h

Description

meiMotorAmpWarning reads the amp warning buffer from a motor and writes the data into a structure pointed to by warning.

motor	a handle to the Motor object
***warning**	a pointer to a structure containing the number of amp warnings, their coded values and message strings. See MEIMotorAmpWarnings.

Return Values	
MPIMessageOK	

See Also

meiMotorAmpWarningClear | meiMotorAmpFault | meiMotorAmpFaultClear

meiMotorAmpWarningClear

Declaration

```
long meiMotorAmpWarningClear(MPIMotor    motor);
```

Required Header: stdmei.h

Description

meiMotorAmpWarningClear flushes the motor's amp warning message buffer. The number of amp warnings is set to zero, the coded values are set to zero, and the messages are cleared.

motor	a handle to the Motor object

Return Values	
MPIMessageOK	

See Also

meiMotorAmpWarning | meiMotorAmpFault | meiMotorAmpFaultClear

mpiMotorAxisMapGet

Declaration

```
long mpiMotorAxisMapGet(MPIMotor      motor,
                        MPIObjectMap  *axismap)
```

Required Header: stdmpi.h

Description

mpiMotorAxisMapGet gets the object map of the Axes associated with a Motor (**motor**) and writes it into the structure pointed to by **axismap**.

motor	a handle to the Motor object
***axismap**	a pointer to an object map. An ObjectMap is a bitmap, where each numbered bit represents the presence or absence of the correspondingly numbered object.

Return Values	
MPIMessageOK	

See Also

meiMotorCommutationModeGet

Declaration

```
long meiMotorCommutationModeGet(MPIMotor       motor,
                                MEIXmpCommMode *mode)
```

Required Header: stdmei.h

Description

meiMotorCommutationModeGet gets the commutation mode of a Motor (***motor***) and writes it to the location pointed to by ***mode***.

Return Values	
MPIMessageOK	

See Also

meiMotorCommutationModeSet

meiMotorCommutationModeSet

Declaration

```
long meiMotorCommutationModeSet(MPIMotor       motor,
                                MEIXmpCommMode mode)
```

Required Header: stdmei.h

Description

meiMotorCommutationModeSet sets the commutation mode of a Motor (***motor***) to ***mode***.

Return Values	
MPIMessageOK	

See Also

meiMotorCommutationModeGet

mpiMotorConfigGet

Declaration

```
long mpiMotorConfigGet(MPIMotor       motor,
                       MPIMotorConfig *config,
                       void           *external)
```

Required Header: stdmpi.h

Description

mpiMotorConfigGet gets a Motor's (***motor***) configuration and writes it into the structure pointed to by ***config***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The configuration information in ***external*** is in addition to the configuration information in ***config***, i.e, the configuration information in ***config*** and in ***external*** is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL).

Remarks

external either points to a structure of type MEIMotorConfig{} or is NULL.

Return Values	
MPIMessageOK	
MEIMotorMessageDEMAND_MODE_NOT_SET	
MEIMotorMessageDEMAND_MODE_UNSUPPORTED	

Sample Code

```
MEIMotorConfig motorConfig;
mpiMotorConfigGet( motor0, NULL, &motorConfig );

motorConfig.Io[0].Type = MEIMotorIoTypeBRAKE;

mpiMotorConfigSet( motor0, NULL, &motorConfig );
```

See Also

MEIMotorConfig | mpiMotorConfigSet

mpiMotorConfigSet

Declaration

```
long mpiMotorConfigSet(MPIMotor       motor,
                       MPIMotorConfig *config,
                       void           *external)
```

Required Header: stdmpi.h

Description

mpiMotorConfigSet sets a Motor's (***motor***) configuration using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The configuration information in ***external*** is *in addition* to the configuration information in ***config***, i.e, the configuration information in ***config*** and in ***external*** is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL).

Remarks

external either points to a structure of type MEIMotorConfig{} or is NULL.

Return Values	
MPIMessageOK	
MEIMotorMessageDEMAND_MODE_NOT_SET	
MEIMotorMessageDEMAND_MODE_UNSUPPORTED	

Sample Code

```
MEIMotorConfig motorConfig;
mpiMotorConfigGet( motor0, NULL, &motorConfig );

motorConfig.Io[0].Type = MEIMotorIoTypeBRAKE;

mpiMotorConfigSet( motor0, NULL, &motorConfig );
```

See Also

mpiMotorConfigGet | MEIMotorConfig
Special Note: Using mpiMotorConfigSet with Absolute Encoders

meiMotorConfigStepper

Declaration

```
long meiMotorConfigStepper(MPIMotor       motor,
                           MEIMotorConfig *config,
                           long           stepperNumber)
```

Required Header: stdmei.h

Description

meiMotorConfigStepper modifies the motor configuration structure pointed to by config, to use a step engine (stepperNumber) from another motor. By default, each motor uses its own step engine. Do NOT use more than one motor per step engine. Use the methods mpiMotorConfigGet/Set(...) to read/write the motor configuration from/to the controller.

motor	a handle to the Motion object
***config**	a pointer to the motion frame buffer status structure returned by the method
stepperNumber	index to a step engine

Return Values	
MPIMessageOK	

See Also

mpiMotorConfigGet | mpiMotorConfigSet

meiMotorDacConfigGet

Declaration

```
long meiMotorDacConfigGet(MPIMotor           motor,
                          MEIMotorDacConfig  *config);
```

Required Header: stdmei.h

Description

meiMotorDacConfigGet gets a Motor's (***motor***) DAC configuration and writes it to the structure pointed to by ***config***.

motor	a handle to the Motor object.
***config**	a pointer to a MEIMotorDacConfig structure.

Return Values	
MPIMessageOK	

See Also

meiMotorDacConfigSet | MEIMotorDacConfig

meiMotorDacConfigSet

Declaration

```
long meiMotorDacConfigSet(MPIMotor          motor,
                          MEIMotorDacConfig *config);
```

Required Header: stdmei.h

Description

meiMotorDacConfigSet configures a Motor's (***motor***) DAC using data from the structure pointed to by ***config***.

motor	a handle to the Motor object.
***config**	a pointer to a MEIMotorDacConfig structure.

Return Values	
MPIMessageOK	

See Also

meiMotorDacConfigGet | MEIMotorDacConfig

meiMotorDacFlashConfigGet

Declaration

```
long meiMotorDacFlashConfigGet(MPIMotor           motor,
                               void               *flash,
                               MEIMotorDacConfig  *config);
```

Required Header: stdmei.h

Description

meiMotorDacFlashConfigGet gets a Motor's (***motor***) DAC configuration from flash memory and writes it to the structure pointed to by ***config***.

motor	a handle to the Motor object
***flash**	***flash*** is either an MEIFlash handle or MPIHandleVOID. If flash is MPIHandleVOID, an MEIFlash object will be created and deleted internally. Using MPIHandleVOID is recommended, as it simplifies code. If ***flash*** is a valid MEIFlash handle, then the MEIFlash object cache will be updated, but the actual write to controller flash will not occur. Use meiFlashMemoryFromFileType(...) to prompt the actual write to flash.
***config**	a pointer to a MEIMotorDacConfig structure.

Return Values	
MPIMessageOK	

See Also

meiMotorDacFlashConfigSet | meiMotorDacConfigGet | MEIMotorDacConfig

meiMotorDacFlashConfigSet

Declaration

```
long meiMotorDacFlashConfigSet(MPIMotor           motor,
                               void               *flash,
                               MEIMotorDacConfig  *config);
```

Required Header: stdmei.h

Description

meiMotorDacFlashConfigSet sets a Motor's (***motor***) DAC configuration to flash memory using data from the structure pointed to by ***config***.

motor	a handle to the Motor object
***flash**	***flash*** is either an MEIFlash handle or MPIHandleVOID. If flash is MPIHandleVOID, an MEIFlash object will be created and deleted internally. Using MPIHandleVOID is recommended, as it simplifies code. If ***flash*** is a valid MEIFlash handle, then the MEIFlash object cache will be updated, but the actual write to controller flash will not occur. Use meiFlashMemoryFromFileType(...) to prompt the actual write to flash.
***config**	a pointer to a MEIMotorDacConfig structure.

Return Values	
MPIMessageOK	

See Also

meiMotorDacFlashConfigGet

mpiMotorDedicatedIn

Declaration

```
long mpiMotorDedicatedIn(MPIMotor        motor,
                         long            startBit,
                         long            bitCount,
                         unsigned long   *state);
```

Required Header: stdmpi.h
Change History: Added in 03.02.00. mpiMotorDedicatedIn replaced mpiMotorIoGet.

Description

mpiMotorDedicatedIn function reads the current state of one or more dedicated input bits.

NOTE: mpiMotorDedicatedIn replaced mpiMotorIoGet in the MPI library.

motor	a handle to the Motor object
startBit	the first dedicated in bit that will be returned by the function.
bitCount	the number of dedicated in bits that will be returned by the function.
***state**	the address of the current state of the inputs that is returned.

Return Values	
MPIMessageOK	

See Also

Dedicated Motor I/O | MPIMotorDedicatedIn

mpiMotorDedicatedOutGet

Declaration

```
long mpiMotorDedicatedOutGet(MPIMotor       motor,
                             long           startBit,
                             long           bitCount,
                             unsigned long  *state);
```

Required Header: stdmpi.h
Change History: Added in 03.02.00

Description

mpiMotorDedicatedOutGet gets the current state of one or more of the dedicated outputs.

motor	a handle to the Motor object
startBit	the first dedicated out bit that will be returned by the function.
bitCount	the number of dedicated out bits that will be returned by the function.
***state**	the address of where the current state of the outputs will be returned.

Return Values	
MPIMessageOK	

See Also

Dedicated Motor I/O | MPIMotorDedicatedOut

mpiMotorFeedback

Declaration

```
long mpiMotorFeedback(MPIMotor           motor,
                      MPIMotorFeedback   *feedback)
```

Required Header: stdmpi.h

Description

mpiMotorFeedback gets the feedback position of a Motor (***motor***) and writes it into the location pointed to by ***feedback***.

Return Values	
MPIMessageOK	

See Also

mpiMotorFlashConfigGet

Declaration

```
long mpiMotorFlashConfigGet(MPIMotor       motor,
                            void           *flash,
                            MPIMotorConfig *config,
                            void           *external)
```

Required Header: stdmpi.h

Description

mpiMotorFlashConfigGet gets a Motor's (***motor***) flash configuration and writes it in the structure pointed to by ***config***, and also writes it in the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The Motor's flash configuration information in ***external*** is in addition to the Motor's flash configuration information in ***config***, i.e, the flash configuration information in ***config*** and in ***external*** is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL).

Remarks

external either points to a structure of type MEIMotorConfig{} or is NULL.

motor	a handle to a Motor object
***flash**	***flash*** is either an MEIFlash handle or MPIHandleVOID. If flash is MPIHandleVOID, an MEIFlash object will be created and deleted internally. Using MPIHandleVOID is recommended, as it simplifies code. If ***flash*** is a valid MEIFlash handle, then the MEIFlash object cache will be updated, but the actual write to controller flash will not occur. Use meiFlashMemoryFromFileType(...) to prompt the actual write to flash.
***config**	a pointer to a configuration structure for the motor object of type MPIMotorConfig.
***external**	a pointer to a configuration structure for the motor object of type MEIMotorConfig.

Return Values	
MPIMessageOK	

See Also

MEIMotorConfig | MEIFlash | mpiMotorFlashConfigSet

mpiMotorFlashConfigSet

Declaration

```
long mpiMotorFlashConfigSet(MPIMotor       motor,
                            void           *flash,
                            MPIMotorConfig *config,
                            void           *external)
```

Required Header: stdmpi.h

Description

mpiMotorFlashConfigSet sets a Motor's (***motor***) flash configuration using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The Motor's flash configuration information in ***external*** is in addition to the Motor's flash configuration information in ***config***, i.e., the flash configuration information in ***config*** and in external is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL).

Remarks

external either points to a structure of type MEIMotorConfig{} or is NULL.

motor	a handle to a Motor object
***flash**	***flash*** is either an MEIFlash handle or MPIHandleVOID. If flash is MPIHandleVOID, an MEIFlash object will be created and deleted internally. Using MPIHandleVOID is recommended, as it simplifies code. If ***flash*** is a valid MEIFlash handle, then the MEIFlash object cache will be updated, but the actual write to controller flash will not occur. Use meiFlashMemoryFromFileType(...) to prompt the actual write to flash.
***config**	a pointer to a configuration structure for the motor object of type MPIMotorConfig.
***external**	a pointer to a configuration structure for the motor object of type MEIMotorConfig.

Return Values	
MPIMessageOK	
MEIFlashMessageNETWORK_TOPOLOGY_ERROR	

See Also

MEIMotorConfig | MEIFlash | mpiMotorFlashConfigGet | meiSynqNetFlashTopologySave

mpiMotorGeneralIn

Declaration

```
long mpiMotorGeneralIn(MPIMotor         motor,
                       long             startBit,
                       long             bitCount,
                       unsigned long    *state);
```

Required Header: stdmpi.h
Change History: Added in 03.02.00. mpiMotorGeneralIn replaced mpiMotorIoGet.

Description

mpiMotorGeneralIn reads the current input state of one or more general purpose bits.

NOTE: mpiMotorGeneralIn replaced mpiMotorIoGet in the MPI library.

motor	a handle to the Motor object
startBit	the first general purpose bit that will be returned by the function.
bitCount	the number of general purpose bits that will be returned by the function.
***state**	the address of the current state of the inputs that is returned.

Return Values	
MPIMessageOK	

Sample Code

```
long x;
meiMotorGeneralIn( motor0, 0, 1, &x );
```

See Also

General Purpose Motor I/O

mpiMotorGeneralOutGet

Declaration

```
long mpiMotorGeneralOutGet(MPIMotor         motor,
                           long             startBit,
                           long             bitCount,
                           unsigned long    *state);
```

Required Header: stdmpi.h
Change History: Added in 03.02.00. mpiMotorGeneralOutGet replaced mpiMotorIoGet.

Description

mpiMotorGeneralOutGet function reads the current output state of one or more general purpose bits.

NOTE: mpiMotorGeneralOutGet replaced mpiMotorIoGet in the MPI library.

motor	a handle to the Motor object
startBit	the first general purpose bit that will be returned by the function.
bitCount	the number of general purpose bits that will be returned by the function.
***state**	the address of the current state of the general purpose bits will be returned.

Return Values	
MPIMessageOK	

Sample Code

```
long x;
meiMotorGeneralOutGet( motor0, 0, MEIMotorGeneralIoLAST, &x );
```

See Also

General Purpose Motor I/O

mpiMotorGeneralOutSet

Declaration

```
long mpiMotorGeneralOutSet(MPIMotor         motor,
                           long             startBit,
                           long             bitCount,
                           unsigned long    state,
                           MPI_BOOL         wait);
```

Required Header: stdmpi.h
Change History: Modified in the 03.03.00
Added in 03.02.00 (mpiMotorGeneralOutSet replaced mpiMotorIoSet).

Description

mpiMotorGeneralOutSet function changes the state of one or more general purpose bits.

NOTE: mpiMotorGeneralOutSet replaced mpiMotorIoSet in the MPI library.

motor	a handle to the Motor object.
startBit	the first general purpose bit that will be set by the function.
bitCount	the number of general purpose bits that will be set by the function.
state	the new state of the general purpose bits.
wait	See Motor Digital Output Waits.

Return Values	
MPIMessageOK	

See Also

General Purpose Motor I/O | Motor Digital Output Waits

meiMotorInfo

Declaration

```
long meiMotorInfo(MPIMotor      motor,
                  MEIMotorInfo  *info);
```

Required Header: stdmei.h

Description

meiMotorInfo reads the static information about the network, node, and drive interface associated with the motor object, and writes it into the structure pointed to by ***info***.

motor	a handle to the Motor object
***info**	a pointer to a motor information structure

Return Values	
MPIMessageOK	

See Also

meiMotorStatus

meiMotorPhaseFindStatus

Declaration

```
long  meiMotorPhaseFindStatus(MPIMotor                    motor,
                              MEIMotorPhaseFindStatus*    status);
```

Required Header: stdmei.h
Change History: Added in the 03.03.00

Description

meiMotorPhaseFindStatus provides the user with information which reflects the status of the drive's Phase Finding Procedure.

motor	a handle to the Motor object.
status	the current state of the phase finding process: in progress, failed, success. It also contains drive specific information pertaining to the current state of the phase finding procedure.

Return Values	
MPIMessageOK	

See Also

meiMotorPhaseFindStart | meiMotorPhaseFindAbort

Motor Phase Finding

mpiMotorStatus / meiMotorStatus

Declaration: mpiMotorStatus

```
long mpiMotorStatus(MPIMotor    motor,
                    MPIStatus   *status,
                    void        *external)
```

Required Header: stdmpi.h

Description

mpiMotorStatus writes a Motor's (***motor***) status into the structure pointed to by ***status***, and also into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The ***motor's*** status information in ***external*** is in addition to the motor's status information in ***status***, i.e, the status configuration information in ***status*** and in ***external*** is not the same information. Note that ***external*** can be NULL (but status must not be NULL).

Remarks

external either points to a structure of type MEIMotorStatus{...} or is NULL.

motor	a handle to the Motor object
***status**	a pointer to the motor status structure returned by the method
***external**	a pointer to an implementation-specific structure

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

Declaration: meiMotorStatus

```
long meiMotorStatus(MPIControl   control,
                    long         motorNumber
                    MPIStatus    *status,
                    void         *external)
```

Required Header: stdmei.h

Description

meiMotorStatus gets a Motor's status and writes it to the structure pointed to by ***status***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The ***motor's*** status information in ***external*** is in addition to the motor's status information in ***status***, i.e, the status configuration information in ***status*** and in ***external*** is not the same information. Note that ***external*** can be NULL (but status must not be NULL).

Remarks

external either points to a structure of type MEIMotorStatus{...} or is NULL.

control	a handle to the Control object
motorNumber	index to the motor
***status**	a pointer to the motor status structure returned by the method
***external**	pointer to an implementation-specific structure

Return Values	
MPIMessageOK	if *MotorStatus* successfully gets the status of a Motor object.
MPIMessageARG_INVALID	if the *status* pointer is NULL.

Sample Code

```
void readDACOutputs(MPIMotor motor)
{
        long returnValue;
        MPIStatus status;
        MEIMotorStatus motorStatus;

   returnValue = mpiMotorStatus(motor, &status, &motorStatus);
   msgCHECK(returnValue);

   printf("Output CMD = %.4f\n", motorStatus.dac.cmd.level);
   printf("Output AUX = %.4f\n", motorStatus.dac.aux.level);
}
```

See Also

MPIStatus | MEIMotorStatus

mpiMotorEventConfigGet

Declaration

```
long mpiMotorEventConfigGet(MPIMotor              motor,
                            MPIEventType          eventType,
                            MPIMotorEventConfig   *src,
                            void                  *external)
```

Required Header: stdmpi.h

Description

mpiMotorEventConfigGet gets the Motor's (***motor***) configuration for the event specified by ***eventType*** and writes it into the structure pointed to by ***eventConfig***, and also writes it to the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The event configuration information in ***external*** is in addition to the event configuration information in ***eventConfig***, i.e, the event configuration information in ***eventConfig*** and in ***external*** is not the same information.

NOTE: Set ***eventConfig*** or ***external*** to NULL. One must be NULL, the other must be passed a pointer.

Remarks

external either points to a structure of type MEIMotorEventConfig{} or is NULL.

Return Values	
MPIMessageOK	

Sample Code

```
for(index = 0; index < AXIS_COUNT; index++)
  { // turn off error limit and limit switch actions for
    motors 0 to AXIS_COUNT
      returnValue = mpiMotorEventConfigGet(motor[index],
         MPIEventTypeLIMIT_ERROR, &eventConfig, NULL);
      msgCHECK(returnValue);

      eventConfig.action = MPIActionNONE;

      returnValue = mpiMotorEventConfigSet(motor[index],
         MPIEventTypeLIMIT_ERROR, &eventConfig, NULL);
      msgCHECK(returnValue);

      returnValue = mpiMotorEventConfigGet(motor[index],
         MPIEventTypeLIMIT_HW_NEG, &eventConfig, NULL);
      msgCHECK(returnValue);

      eventConfig.action = MPIActionNONE;

      returnValue = mpiMotorEventConfigSet(motor[index],
         MPIEventTypeLIMIT_HW_NEG, &eventConfig, NULL);
      msgCHECK(returnValue);

      returnValue = mpiMotorEventConfigGet(motor[index],
         MPIEventTypeLIMIT_HW_POS, &eventConfig, NULL);
      msgCHECK(returnValue);

      eventConfig.action = MPIActionNONE;

      returnValue = mpiMotorEventConfigSet(motor[index],
         MPIEventTypeLIMIT_HW_POS, &eventConfig, NULL);
      msgCHECK(returnValue);
}
```

See Also

MEIMotorEventConfig | mpiMotorEventConfigSet | Error Limit and Limit Switch Errors

mpiMotorEventConfigSet

Declaration

```
long mpiMotorEventConfigSet(MPIMotor            motor,
                            MPIEventType        eventType,
                            MPIMotorEventConfig *eventConfig,
                            void                *external)
```

Required Header: stdmpi.h

Description

mpiMotorEventConfigSet reads the structure pointed to by ***eventConfig*** and sets the Motor's (***motor***) configuration for the event specified by ***eventType***.

The event configuration information in ***external*** is in addition to the event configuration information in ***eventConfig***, i.e, the event configuration information in ***eventConfig*** and in ***external*** is not the same information.

NOTE: Set ***eventConfig*** or ***external*** to NULL. One must be NULL, the other must be passed a pointer.

Remarks

external either points to a structure of type MEIMotorEventConfig{} or is NULL.

Return Values	
MPIMessageOK	

See Also

MEIMotorEventConfig | mpiMotorEventConfigGet | Error Limit and Limit Switch Errors

mpiMotorEventNotifyGet

Declaration

```
long mpiMotorEventNotifyGet(MPIMotor      motor,
                            MPIEventMask  *eventMask,
                            void          *external)
```

Required Header: stdmpi.h

Description

mpiMotorEventNotifyGet writes the event mask (that specifies the event type(s) for which host notification has been requested) to the location pointed to by ***eventMask***, and also writes it into the implementation-specific location pointed to by ***external*** (if ***external*** is not NULL).

The event notification information in ***external*** is in addition to the event notification information in ***eventmask***, i.e, the event notification information in ***eventmask*** and in ***external*** is not the same information. Note that ***eventmask*** or ***external*** can be NULL (but not both NULL).

Event notification is enabled for event types specified in ***eventmask***, which is a bit mask of MPIEventMask bits associated with the desired MPIEventType values. Event notification is disabled for event types not specified in ***eventmask***. The MPIEventMask bits must be set of cleared using the MPIEventMask macros.

Remarks

external either points to a structure of type MEIEventNotifyData{} or is NULL. The MEIEventNotifyData {} structure is an array of firmware addresses, whose contents are placed into the MEIEventStatusInfo {} structure (of all events generated by this object).

Return Values	
MPIMessageOK	

See Also

MPIEventType | MEIEventNotifyData | MEIEventStatusInfo | mpiMotorEventNotifySet

mpiMotorEventNotifySet

Declaration

```
long mpiMotorEventNotifySet(MPIMotor     motor,
                            MPIEventMask eventMask,
                            void         *external)
```

Required Header: stdmpi.h

Description

mpiMotorEventNotifySet requests host notification of the event(s) that are generated by ***motor*** and specified by ***eventMask***, and also specified by the implementation-specific location pointed to by ***external*** (if ***external*** is not NULL).

The event notification information in ***external*** is in addition to the event notification information in ***eventmask***, i.e, the event notification information in ***eventmask*** and in ***external*** is not the same information. Note that ***eventmask*** or ***external*** can be NULL (but not both NULL).

Event notification is enabled for event types specified in ***eventMask***, a bit mask of MPIEventMask bits associated with the desired MPIEventType values. Event notification is disabled for event types that are not specified in ***eventMask***. The MPIEventMask bits must be set of cleared using the MPIEventMask macros.

The mask of event types generated by a Motor object consists of bits from MPIEventMaskMOTION and MPIEventMaskAXIS.

Remarks

external either points to a structure of type MEIEventNotifyData{} or is NULL. The MEIEventNotifyData {} structure is an array of firmware addresses, whose contents are placed into the MEIEventStatusInfo {} structure (of all events generated by this object).

To	Then
enable host notification of all events	set ***eventmask*** to MPIEventMaskALL
disable host notification of all events	set ***eventmask*** to MPIEventTypeNONE

Return Values	
MPIMessageOK	

Sample Code

```
MPIEventMask eventMask;

mpiEventMaskCLEAR( eventMask );
mpiEventMaskALL( eventMask );
meiEventMaskALL( eventMask );

returnValue = mpiMotorEventNotifySet( motor, eventMask, NULL );
msgCHECK( returnValue );
```

See Also

MPIEventType | MEIEventNotifyData | MEIEventStatusInfo | mpiMotorEventNotifyGet

mpiMotorEventReset

Declaration

```
long mpiMotorEventReset(MPIMotor      motor,
                        MPIEventMask  eventMask)
```

Required Header: stdmpi.h

Description

mpiMotorEventReset resets the event(s) that are specified in ***eventMask*** and generated by ***motor***. Your application must call *MotorEventReset* only after one or more latchable events have occurred.

Return Values	
MPIMessageOK	

See Also

mpiControlEventReset | mpiMotionEventReset | mpiRecorderEventReset | mpiSequenceEventReset | meiSynqNetEventReset | meiSqNodeEventReset | mpiAxisEventReset

Event Notification Methods

mpiMotorMemory

Declaration

```
long mpiMotorMemory(MPIMotor  motor,
                    void      **memory)
```

Required Header: stdmpi.h

Description

mpiMotorMemory sets (writes) an address (used to access a Control object's memory) to the contents of ***memory***.

Return Values	
MPIMessageOK	

See Also

mpiMotorMemoryGet | mpiMotorMemorySet

mpiMotorMemoryGet

Declaration

```
long mpiMotorMemoryGet(MPIMotor      motor,
                       void          *dst,
                       const void    *src,
                       long          count)
```

Required Header: stdmpi.h
Change History: Modified in the 03.03.00

Description

mpiMotorMemoryGet copies ***count*** bytes of a Motor's (***motor***) memory (starting at address ***src***) to application memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

See Also

mpiMotorMemorySet | mpiMotorMemory

mpiMotorMemorySet

Declaration

```
long mpiMotorMemorySet(MPIMotor    motor,
                       void        *dst,
                       const void  *src,
                       long        count)
```

Required Header: stdmpi.h
Change History: Modified in the 03.03.00

Description

mpiMotorMemorySet copies ***count*** bytes of application memory (starting at address ***src***) to a Motor's (***motor***) memory (starting at address ***dst***).

Return Values	
MPIMessageOK	

See Also

mpiMotorMemoryGet | mpiMotorMemory

meiMotorEncoderReset

Declaration

```
long meiMotorEncoderReset(MPIMotor motor)
```

Required Header: stdmei.h

Description

meiMotorEncoderReset clears the encoder fault status registers for the primary and secondary encoder associated with the motor object.

Return Values	
MPIMessageOK	

See Also

meiMotorMultiTurnReset

Declaration

```
long meiMotorMultiTurnReset(MPIMotor    motor);
```

Required Header: stdmei.h

Description

meiMotorMultiTurnReset clears the SynqNet drive multi-turn data for absolute type encoders. This is only needed when configuring the zero location for an absolute encoder or when the absolute encoder's battery is replaced. meiMotorMultiTurnReset is an offline operation. Make sure all motors (amp enables) are disabled before executing a multi-turn reset. The SynqNet network may be shutdown (dropped from SYNQ mode) due to specific drive limitations.

Not all SynqNet drives support or require this feature. Please see the drive manufacturer's documentation for details.

motor	a handle to the Motor object

Return Values	
MPIMessageOK	

See Also

meiMotorPhaseFindAbort

Declaration

```
long meiMotorPhaseFindAbort(MPIMotor   motor);
```

Required Header: stdmei.h
Change History: Added in the 03.03.00

Description

meiMotorPhaseFindAbort stops the phase finding process and disables the amplifier.

motor	a handle to the Motor object.

Return Values	
MPIMessageOK	

See Also

meiMotorPhaseFindStart | meiMotorPhaseFindStatus

Motor Phase Finding

meiMotorPhaseFindStart

Declaration

```
long meiMotorPhaseFindStart(MPIMotor   motor);
```

Required Header: stdmei.h
Change History: Added in the 03.03.00

Description

meiMotorPhaseFindStart activates a drive's phase finding process.

motor	a handle to the Motor object.

Return Values	
MPIMessageOK	

See Also

meiMotorPhaseFindAbort | meiMotorPhaseFindStatus

Motor Phase Finding

mpiMotorControl

Declaration

```
const MPIControl mpiMotorControl(MPIMotor motor)
```

Required Header: stdmpi.h

Description

mpiMotorControl returns a handle to the Control object with which the motor is associated.

motor	a handle to the Motor object

Return Values	
MPIControl	handle to a Control object
MPIHandleVOID	if motor is invalid

See Also

mpiMotorCreate | mpiControlCreate

mpiMotorFilterMapGet

Declaration

```
long mpiMotorFilterMapGet(MPIMotor     motor,
                          MPIObjectMap *filtermap)
```

Required Header: stdmpi.h

Description

mpiMotorFilterMapGet gets the object map of the Filters [that are associated with a Motor (***motor***)] and writes it into the structure pointed to by ***filtermap***.

Return Values	
MPIMessageOK	

See Also

mpiMotorFilterMapSet

mpiMotorFilterMapSet

Declaration

```
long mpiMotorFilterMapSet(MPIMotor     motor,
                          MPIObjectMap filtermap)
```

Required Header: stdmpi.h

Description

mpiMotorFilterMapSet sets the Filters [that are associated with a Motor (***motor***)], using data from the object map specified by ***filtermap***.

Return Values	
MPIMessageOK	

See Also

mpiMotorFilterMapGet

mpiMotorNumber

Declaration

```
long mpiMotorNumber(MPIMotor motor,
                    long     *number)
```

Required Header: stdmpi.h

Description

mpiMotorNumber writes the index of a Motor (***motor***, on the motion controller that ***motor*** is associated with) to the contents of ***number***.

Return Values	
MPIMessageOK	

See Also

meiMotorEncoderRatio

Declaration

```
long meiMotorEncoderRatio(MPIControl           control,
                          long                 motorNumber,
                          long                 encoderNumber,
                          MEIMotorEncoderRatio *ratio)
```

Required Header: stdmei.h

Description

meiMotorEncoderRatio gets encoder ratio from the XMP.

WARNING: This is a customer-specific method that is only supported with custom firmware. To inquire about using this method, please contact an MEI Applications Engineer.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

MEIMotorAmpFaults

Definition

```
typedef struct MEIMotorAmpFaults {
    long                 count;
    long                 code[MEIMotorAmpFaultsMAX];
    MEIMotorAmpFaultMsg  message[MEIMotorAmpFaultsMAX];
} MEIMotorAmpFaults;
```

Description

MPIAxisInPosition contains the amp fault information from a SynqNet drive. The amp fault messages are drive specific. Not all drives support amp fault messages. For details, please see the SqNodeLib header files and the drive manufacturer's documentation.

count	The number of amp faults in the buffer.
code	An array of drive specific amp fault coded values.
message	An array of drive specific amp fault message strings.

See Also

meiMotorAmpFault | meiMotorAmpFaultClear

MEIMotorAmpFaultsMsg

Definition

```
typedef char   MEIMotorAmpFaultMsg[MEIMotorAmpMsgMAX];
```

Description

MEIMotorAmpFaultsMsg defines the amp fault message string definition.

See Also

MEIMotorAmpMsgMAX

MEIMotorAmpWarnings

Definition

```
typedef struct MEIMotorAmpWarnings {
    long                    count;
    long                    code[MEIMotorAmpWarningsMAX];
    MEIMotorAmpWarningMsg   message[MEIMotorAmpWarningsMAX];
} MEIMotorAmpWarnings;
```

Description

MEIMotorAmpWarnings contains the amp warning information from a SynqNet drive. The amp warning messages are drive specific. Not all drives support amp warning messages. For details, please see the SqNodeLib header files and the drive manufacturer's documentation.

count	The number of amp faults in the buffer.
code	An array of drive specific amp fault coded values.
message	An array of drive specific amp fault message strings.

See Also

meiMotorAmpWarning | meiMotorAmpWarningClear

MEIMotorAmpWarningsMsg

Definition

```
typedef  char  MEIMotorAmpWarningsMsg[MEIMotorAmpMsgMAX];
```

Description

MEIMotorAmpWarningsMsg defines the maximum number of amp warning messages per motor.

See Also

MPIMotorBrake

Definition

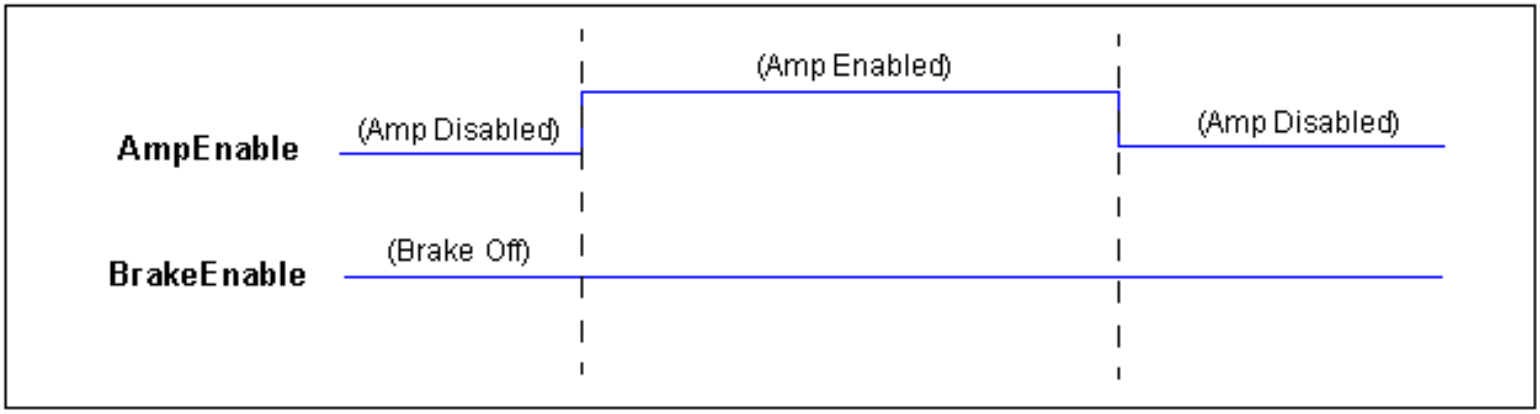
```
typedef struct MPIMotorBrake {
    MPIMotorBrakeMode   mode;
    float               applyDelay;
    float               releaseDelay;
} MPIMotorBrake;
```

Description

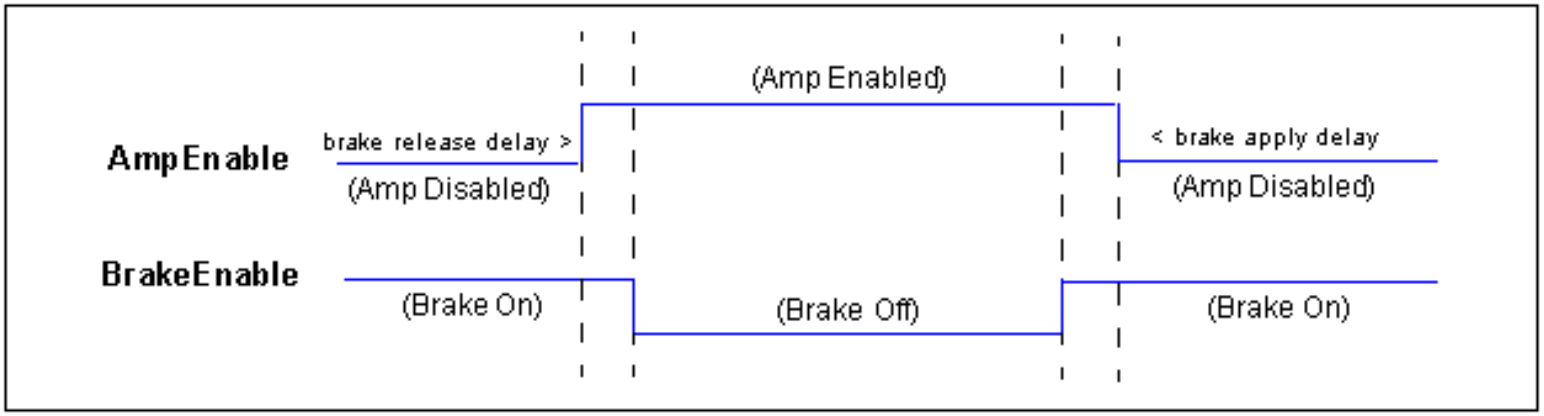
MPIMotorBrake specifies the configuration for a motor's dedicated brake logic. Each motor object has a dedicated brake output. The controller enables/disables the brake depending on the amp enable state and the brake configuration. When the amp enable is disabled, the brake is set to an active state. When the amp enable is enabled, the brake is set to an inactive state.

mode	An enumerated brake mode. See MPIMotorBrakeMode.
applyDelay	The time between when the brake is active and the amp enable is disabled. The units are in seconds.
releaseDelay	The time between when the amp enable is enabled and the brake is inactive. The units are in seconds.

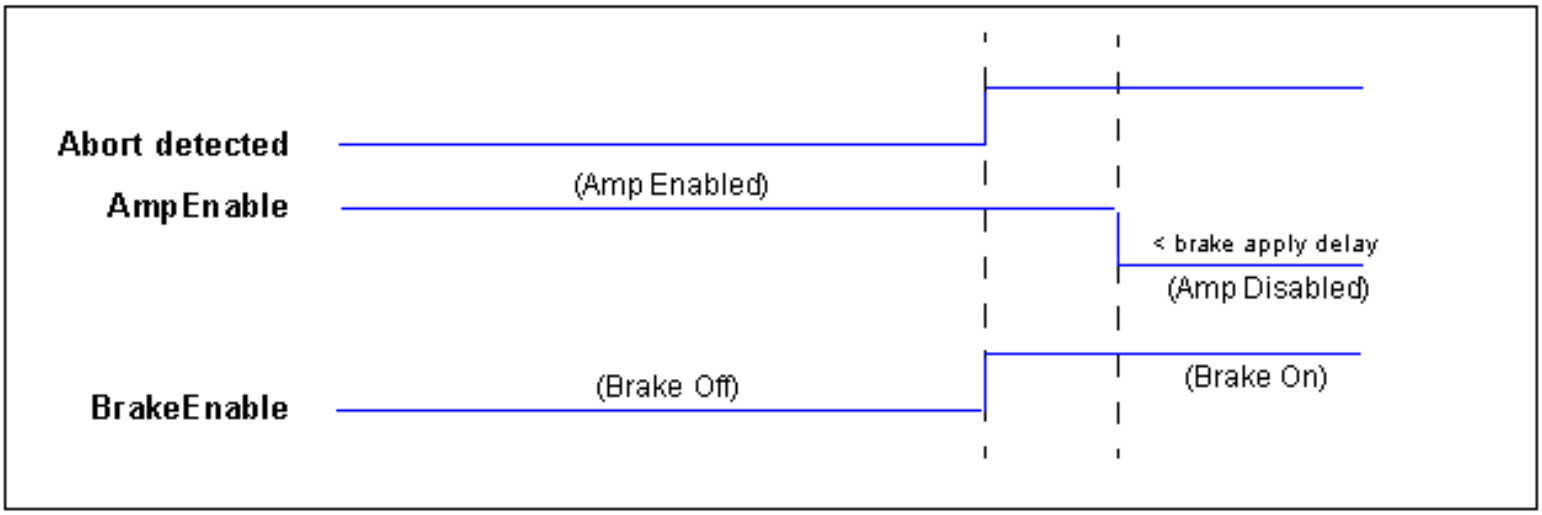
See the diagrams below for the brake logic details:



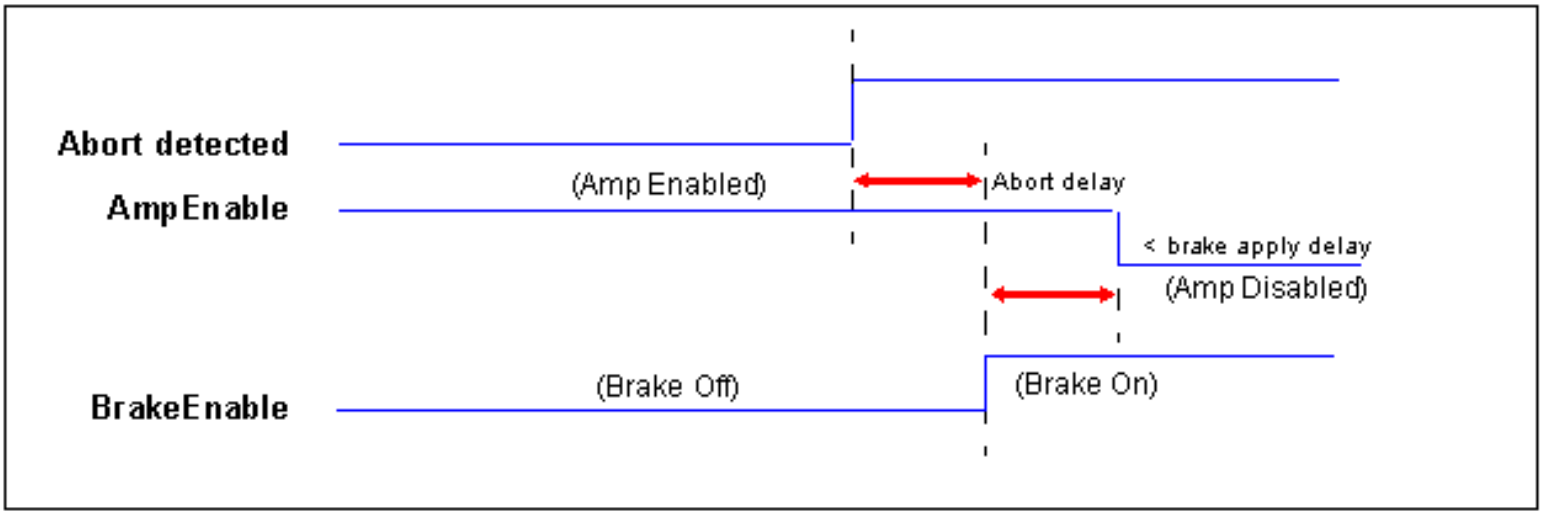
Case 1. Amp On/Off with NO Brake



Case 2. Amp On/Off with Brake



Case 3. Amp Off after ABORT Detection (in system with NO abort delay)



Case 4. Amp Off after ABORT Detection (in system WITH abort delay)

See Also

mpiMotorConfigGet | mpiMotorConfigSet | MPIMotorConfig | MPIMotorDedicatedOut

MPIMotorBrakeMode

Definition

```
typedef enum{
    MPIMotorBrakeModeNONE,
    MPIMotorBrakeModeDELAY,
} MPIMotorBrakeMode;
```

Description

MPIMotorBrakeMode is an enumeration of modes for the dedicated brake signal.

MPIMotorBrakeModeNONE	Brake feature is disabled.
MPIMotorBrakeModeDELAY	Brake is enabled/disabled with specified delays.

See Also

MPIMotorBrake | MPIMotorConfig

MPIMotorConfig / MEIMotorConfig

Definition: MPIMotorConfig

```
typedef struct MPIMotorConfig {
    MPIMotorType     type;

    /* Event configuration, ordered by MPIEventType */
    MPIMotorEventConfig    event[MPIEventTypeMOTOR_LAST];

    float           abortDelay;
    float           enableDelay;
    MPIMotorBrake   brake;

    MPIObjectMap    filterMap;

} MPIMotorConfig;
```

Description

event	Structure to configure various Motor Events. See MPIMotorEventConfig description.
abortDelay	Sets time value, in seconds, to delay Abort action after Event has occured.
enableDelay	Sets time value, in seconds, to delay Enabling of the amplifier after commanded.
brake	Configures the dedicated brake logic. See MPIMotorBrake.
filterMap	Get/Set a map of Filter Objects to which the Motor is mapped. Default mapping is Filter 0 to Motor 0, Filter 1 to Motor 1, etc. See also MPIObjectMap description in Object section.

Definition: MEIMotorConfig

```
typedef struct MEIMotorConfig {
    char                    userLabel[MEIObjectLabelCharMAX+1];
                              /* +1 for NULL terminator */
    MEIMotorDemandMode      demandMode;
    MEIMotorEncoder         Encoder[MEIXmpMotorEncoders];
    MEIMotorStatusOutput    StatusOutput;

    MEIMotorIoConfig        Io[MEIMotorIoConfigIndexLAST];

    MEIMotorFaultConfig     faultConfig;

    MEIMotorStepper         Stepper;
    MEIMotorDacConfig       Dac;

    MEIXmpCommutationBlock  Commutation;
                              /* read-only from field Theta to end */

    MEIXmpLimitData         Limit[MEIXmpLimitLAST];

    MPIAction               nodeFailureAction;
    MPIAction               userFaultAction;
                              /* see MEISqNodeConfigUserFault{}
                              structure */
    MEIMotorDisableAction   disableAction;

} MEIMotorConfig;
```

Required Header: stdmei.h
Change History: Modified in the 03.04.00. Modified in the 03.03.00.

Description

MEIMotorConfig contains configurations for the motor.

userLabel	This field consists of 16 characters and is used to label the motot object for user identification purposes. The userLabel field is NOT used by the controller.
demandMode	The demand mode determines the data type(s) transmitted from the controller to the node for servo control. The default value is automatically configured during SynqNet network initialization, based on the particular node model. The demand mode cannot be changed when the motor's **ampEnable** is enabled for safety. See MEIMotorDemandMode description.
Encoder	Structure to configure Motor Encoder type and parameters.
StatusOutput	A structure to configure a motor's digital outputs to monitor axis status bits. Requires custom firmware.
Io	An array of motor I/O configuration structures.
faultConfig	A structure to configure a motor's fault bits. Support for motor fault bits is node/drive specific.
Stepper	Structure to configure Motor Stepper parameters. See MEIMotorStepper description.
Dac	Structure that includes Command and Auxiliary DAC configuration for each motor. See MEIMotorDacConfig description.
Commutation	A structure to configure controller sinusoidal commutation. This structure is controller specific. Please see Sinusoidal Commutation for more details.
Limit	Structure used to configure custom motor limits and events. See User Limits for more information.
nodeFailureAction	Action applied to the motor when a Node failure occurs.
userFaultAction	Action applied to the motor when a User Fault occurs.
disableAction	Used to configure the controller to set the command position equal to the actual position while the motor is disabled.

See Also

mpiMotorConfigGet | mpiMotorConfigSet

MEIMotorDacConfig

Definition

```
typedef struct MEIMotorDacConfig {
    MEIXmpDACPhase              Phase;
    MEIMotorDacChannelConfig    Cmd;
    MEIMotorDacChannelConfig    Aux;
} MEIMotorDacConfig;
```

Description

MEIMotorDacConfig is a structure that includes Command and Auxiliary DAC configuration for each motor.

See Also

meiMotorDacConfigGet | meiMotorDacConfigSet

MEIMotorDacChannelConfig

Definition

```
typedef struct MEIMotorDacChannelConfig {
    float               Offset; /* volts */
    float               Scale;
    MEIXmpDACInputType  InputType;
    MEIXmpGenericValue  *Input;
} MEIMotorDacChannelConfig;
```

Description

MEIMotorDacChannelConfig is a structure used to configure the DAC settings.

Offset	Set DAC Offset value. Valid values range from -10 Volts to +10 Volts.
Scale	Multiplier for the Dac channel. Default value is 1.0.
InputType	An enumerated value to define whether the Dac channel input is a float or a long. The input type is reserved for special or custom configurations.
Input	A pointer to a MEIXmpGenericValue, which is a union of a long and float value. The input pointer is reserved for special or custom configurations.

See Also

MEIMotorDacChannelStatus

Definition

```
typedef struct MEIMotorDacChannelStatus {
    float       level;  /* volts */
} MEIMotorDacChannelStatus;
```

Description

MEIMotorDacChannelConfig is a structure used to configure the DAC settings.

level	*level* reflects the DAC output value. Valid values range from -10 Volts to +10 Volts.

See Also

MEIMotorDacStatus

Definition

```
typedef struct MEIMotorDacStatus {
    MEIMotorDacChannelStatus    cmd;
    MEIMotorDacChannelStatus    aux;
} MEIMotorDacStatus;
```

Description

MEIMotorDacStatus is a structure that returns the Status for both Command and Auxiliary DACs. It is used to read the ***cmd*** and ***aux*** DAC level (in volts) from the controller.

See Also

MPIMotorDedicatedIn

Definition

```
typedef enum {
   MPIMotorDedicatedInAMP_FAULT                =  0,
   MPIMotorDedicatedInBRAKE_APPLIED            =  1,
   MPIMotorDedicatedInHOME                     =  2,
   MPIMotorDedicatedInLIMIT_HW_POS             =  3,
   MPIMotorDedicatedInLIMIT_HW_NEG             =  4,
   MPIMotorDedicatedInINDEX_PRIMARY            =  5,
   MPIMotorDedicatedInFEEDBACK_FAULT           =  6,
   MPIMotorDedicatedInCAPTURED                 =  7,
   MPIMotorDedicatedInHALL_A                   =  8,
   MPIMotorDedicatedInHALL_B                   =  9,
   MPIMotorDedicatedInHALL_C                   =  10,
   MPIMotorDedicatedInAMP_ACTIVE               =  11,
   MPIMotorDedicatedInINDEX_SECONDARY          =  12,
   MPIMotorDedicatedInWARNING                  =  13,
   MPIMotorDedicatedInDRIVE_STATUS_9           =  14,
   MPIMotorDedicatedInDRIVE_STATUS_10          =  15,
   MPIMotorDedicatedInFEEDBACK_FAULT_PRIMARY   =  16,
   MPIMotorDedicatedInFEEDBACK_FAULT_SECONDARY =  17,
} MPIMotorDedicatedIn;
```

Change History: Modified in the 03.03.00
Added in the 03.02.00 (MPIMotorDedicatedIn replaced MEIMotorDedicatedIn).

Description

MPIMotorDedicatedIn is an enumeration of bit masks for the motor's dedicated inputs. The support for dedicated inputs is node/drive specific. See the node/drive manufacturer's documentation for details.

NOTE: MPIMotorDedicatedIn replaced MEIMotorDedicatedIn in the MPI library.

MPIMotorDedicatedInAMP_FAULT	Generated by the masked motor fault bits. Active when one or more masked motor faults bits are active. See MEIMotorFaultConfig.
MPIMotorDedicatedInBRAKE_APPLIED	Mechanical brake state.
MPIMotorDedicatedInHOME	Position calibration sensor.
MPIMotorDedicatedInLIMIT_HW_POS	Hardware limit for the positive direction.
MPIMotorDedicatedInLIMIT_HW_NEG	Hardware limit for the negative direction.
MPIMotorDedicatedInINDEX_PRIMARY	Primary encoder index input signal.
MPIMotorDedicatedInFEEDBACK_FAULT	Position feedback status. TRUE when position feedback fails, FALSE when operating properly.
MPIMotorDedicatedInCAPTURED	Currently not supported.
MPIMotorDedicatedInHALL_A	Reflects the state of Hall Sensor A
MPIMotorDedicatedInHALL_B	Reflects the state of Hall Sensor B
MPIMotorDedicatedInHALL_C	Reflects the state of Hall Sensor C
MPIMotorDedicatedInAMP_ACTIVE	A bit set by the drive that indicates the amplifier's state. 1 = Amplifier is closing the current loop and the motor winding are energized. 0 = Amplifier is not closing the current loop and the motor windings are not energized. Support for this bit varies depending on the drive type.
MPIMotorDedicatedInINDEX_SECONDARY	Secondary encoder index input signal.

MPIMotorDedicatedInWARNING	Drive warning state. 1 = drive warning status bit is active and warning message is available 0 = drive warning status bit is not active. Support for this bit varies depending on the drive type.
MPIMotorDedicatedInDRIVE_STATUS_9	State of bit 9 in the SynqNet drive specific status register.
MPIMotorDedicatedInDRIVE_STATUS_10	State of bit 10 in the SynqNet drive specific status register.
MPIMotorDedicatedInFEEDBACK_FAULT_PRIMARY	Indicates that the drive/motor primary position feedback system has detected a fault.
MPIMotorDedicatedInFEEDBACK_FAULT_SECONDARY	Indicates that the drive/motor secondary position feedback system has detected a fault.

See Also

MPIMotorDedicatedOut | mpiMotorDedicatedIn

MPIMotorDedicatedOut

Definition

```
typedef enum {
    MPIMotorDedicatedOutAMP_ENABLE
          = MEIXmpMotorDedicatedFlagsMaskAMP_ENABLE,    /* bit 0 */
    MPIMotorDedicatedOutBRAKE_RELEASE
          = MEIXmpMotorDedicatedFlagsMaskBRAKE_RELEASE, /* bit 1 */
} MPIMotorDedicatedOut;
```

Change History: Added in the 03.02.00. MPIMotorDedicatedOut replaced MEIMotorDedicatedOut.

Description

MPIMotorDedicatedOut is an enumeration of bit masks for the motor's dedicated outputs. The support for dedicated outputs is node/drive specific. See the node/drive manufacturer's documentation for details.

NOTE: MPIMotorDedicatedOut replaced MEIMotorDedicatedOut in the MPI library.

MPIMotorDedicatedOutAMP_ENABLE	Enable/disable drive or amplifier. Drive is enabled when TRUE, disabled when FALSE.
MPIMotorDedicatedOutBRAKE_RELEASE	Enable/disable mechanical brake. Brake is released (motor shaft is free) when TRUE, engaged when FALSE.

See Also

MPIMotorDedicatedIn | mpiMotorDedicatedOutGet

MEIMotorDemandMode

Definition

```
typedef enum MEIMotorDemandMode {
   MEIMotorDemandModeTORQUE,
   MEIMotorDemandModeVELOCITY,
   MEIMotorDemandModeANALOG,
   MEIMotorDemandModeANALOG_DUAL_DAC,
} MEIMotorDemandMode;
```

Change History: Added in the 03.04.00

Description

MEIMotorDemandMode is an enumeration of demand modes for a motor. The demand mode determines the data type(s) transmitted from the controller to the node for servo control.

During SynqNet network initialization, the nodes are discovered and the demand mode is automatically set to a default based on the particular drive model. The number of demand fields (1, 2, or 3) per motor are enabled and connected to the controller's filter (closed-loop servo algorithm). The user can change the demand mode using mpiMotorConfigGet/Set(…).

Please consult the drive or RMB (Remote Motion Block) documentation to determine which modes are supported.

MEIMotorDemandModeTORQUE	The controller sends a 16-bit motor control value representing torque. For SynqNet drives only.
MEIMotorDemandModeVELOCITY	The controller sends a 16-bit motor control value representing velocity. For SynqNet drives only.
MEIMotorDemandModeANALOG	The controller sends a 16-bit motor control value representing torque. For RMBs only.
MEIMotorDemandModeANALOG_DUAL_DAC	The controller sends two 16-bit motor control values. Use this mode for auxiliary DACs or sinusoidal commutation. For RMBs only.

Remarks

For optimum controller performance with RMBs, set **demandMode** = MEIMotorDemandModeANALOG_DUAL_DAC ONLY for motors that actually use the auxiliary DACs.

Sample Code

To configure the SynqNet demand mode for velocity:

```
MEIMotorConfig config;
mpiMotorConfigGet(motor, NULL, &config);

config.demandMode = MEIMotorDemandModeVELOCITY;

mpiMotorConfigSet(motor, NULL, &config);
```

See Also

MEIMotorConfig | mpiMotorConfigSet | mpiMotorConfigGet

SynqNet Demand Modes

MEIMotorDisableAction

Definition

```
typedef enum MEIMotorDisableAction {
    MEIMotorDisableActionNONE,
    MEIMotorDisableActionCMD_EQ_ACT,
} MEIMotorDisableAction;
```

Description

MEIMotorDisableAction is an enumeration of controller actions to be applied when the Amp Enable is disabled. This feature can be configured by calling mpiMotorConfigSet(...) with the disableAction element in the MEIMotorConfig structure set to one of the values defined in the MEIMotorDisableAction enumeration.

The default configuration is MEIMotorDisableActionCMD_EQ_ACT. This configuration applies some safety features which eliminate motor jumps when the Amp Enable is enabled. This is the recommended configuration for all servo motor types.

The CMD_EQ_ACT action does not operate with stepper motors. If the motor type is a stepper, make sure to set the disable action to NONE. The pulse output is based on the command position, so if the controller sets the command position equal to the actual position during motion, it will cause very unusual motion profiles.

MEIMotorDisableActionNONE	No action. When the Amp Enable is disabled (or enabled), the controller continues to calculate and apply the torque demand output value.
MEIMotorDisableActionCMD_EQ_ACT	Command position equals actual position action (default). When the Amp Enable is disabled, the controller will: 1) Disable the servo loop output (except for the offset). 2) Set the command position equal to the actual position every sample. 3) Clear the integrator error. When the Amp Enable is enabled, the controller will operate the servo loop normally.

See Also

mpiMotorConfigSet | MEIMotorConfig | MEIMotorDisableAction

MPIMotorEncoder / MEIMotorEncoder

Definition: MPIMotorEncoder

```
typedef enum {
    MPIMotorEncoderPRIMARY,
    MPIMotorEncoderSECONDARY,
} MPIMotorEncoder;
```

Description

MPIMotorEncoder is an enumeration of encoder feedback inputs for a motor.

MPIMotorEncoderPRIMARY	The first encoder feedback for a motor.
MPIMotorEncoderSECONDARY	The second encoder feedback for a motor.

Definition: MEIMotorEncoder

```
typedef struct MEIMotorEncoder {
    MEIMotorEncoderType             type;
    long                            encoderPhase;
                                    /* 0 => normal, else reversed */
    long                            filterDisable;
                                    /* 0 => quad filter enabled,
                                    else not enabled */
    MEIMotorEncoderRatio            ratio;
    MEIMotorEncoderModulo           modulo;
    MEIMotorEncoderSsiConfig        ssiConfig;
} MEIMotorEncoder;
```

Required Header: stdmei.h
Change History: Modified in the 03.04.00

Description

MEIMotorEncoder is an enumeration of encoder feedback inputs for a motor.

For **standard modulo**, set modulo.value to the modulo value and set modulo.reverse to FALSE.

For **reverse modulo**, set modulo.value to the modulo value from the raw encoder and set modulo.reverse to TRUE.

type†	The type of feedback being used by the motor.
encoderPhase†	Controls the direction of the encoder counts (only applicable with quadrature encoder feedback).
filterDisable†	Enables/Disables the use of filters on the encoder signal.
ratio	Custom firmware required. Encoder feedback ratio for scaling.
modulo	Custom firmware required. Reverse modulo value for the encoder to handle rollover.
ssiConfig	A structure to configure the motor feedback for SSI (Synchronous-Serial Interface). Only use **ssiConfig** when the MEIMotorEncoderType is configured for **MEIMotorEncoderTypeSSI**.

† - If you attempt to change this value and hardware is not connected, an error message will be returned.

See Also

Error Limit and Limit Switch Errors | mpiMotorCreate | MEIMotorEncoderType

MPIMotorEncoderFault

Definition

```
typedef enum {
    MPIMotorEncoderFaultPRIMARY,
    MPIMotorEncoderFaultSECONDARY,
    MPIMotorEncoderFaultPRIMARY_OR_SECONDARY,
} MPIMotorEncoderFault;
```

Description

MPIMotorEncoderFault is an enumeration of encoder fault sources for motor encoder fault events. Each motor object supports a primary and secondary encoder. The hardware may or may not support a secondary encoder.

MPIMotorEncoderFaultPRIMARY	Sets the Motor Event (Encoder Fault) to trigger from the primary encoder.
MPIMotorEncoderFaultSECONDARY	Sets the Motor Event (Encoder Fault) to trigger from the secondary encoder.
MPIMotorEncoderFaultPRIMARY_OR_SECONDARY	Sets the Motor Event (Encoder Fault) to trigger from either the primary or secondary encoder.

See Also

MPIMotorEventConfig | MPIMotorEventTrigger

MPIMotorEncoderFaultMask

Definition

```
typedef enum {
     MPIMotorEncoderFaultMaskNONE,
     MPIMotorEncoderFaultMaskBW_DET,
     MPIMotorEncoderFaultMaskILL_DET,
     MPIMotorEncoderFaultMaskABS_ERR,
     MPIMotorEncoderFaultMaskALL
} MPIMotorEncoderFaultMask;
```

Description

MPIMotorEncoderFaultMask is an enumeration to mask bits from MPIMotorEncoderFault register.

MPIMotorEncoderFaultMaskBW_DET	Mask for Broken Wire detection
MPIMotorEncoderFaultMaskILL_DET	Mask for Illegal State detection
MPIMotorEncoderFaultMaskABS_ERR	Mask for Absolute Encoder Error

See Also

MEIMotorEncoderModulo

Definition

```
typedef struct MEIMotorEncoderModulo {
   MPI_BOOL   reverse;  /* set to TRUE if external encoder is
                           already moduloed at a non 32 bit boundary */
   long       value;
} MEIMotorEncoderModulo;
```

Change History: Added in the 03.04.00.

Description

The controller can modulo incomming encoder feedback. In cases where the you want to keep track of how many times the motor has passed a certain position (ex: how many times a motor has gone through one revolution), the modulo function can be used. If you want to keep track of how many times a motor has gone through one revolution, set the modulo ***value*** to the number of encoder counts per revolution and set ***reverse*** to FALSE. The lower 32 bits of the motor feedback will represent the position within one revolution of the motor. The upper 32 bits will represent how many times the motor has gone through one revolution.

For motors that do not have 32 bits of feedback, the reverse modulo function MUST be used. Set the modulo ***value*** equal to the encoder resolution and set ***reverse*** to TRUE. The controller will create a 64-bit encoder position in the motor object from the encoder feedback.

reverse	If the encoder provides 32 bits of feedback, set the reverse field to FALSE. In the case where an encoder does not provide 32 bits of feedback, the reverse field is used. For example, if an encoder has 24 bits of feedback, set the modulo value to 2^{24} and set the reverse field to TRUE. The controller will create a full 64-bit encoder position based on the 24-bit feedback from the motor.
value	Set to the Modulo value. Modulo is always active. The modulo value can be any number from 0 to 4294967295 (2^{32}-1). A value of 0 (default) is equivalent to 2^{32} (0x100000000) and causes a rollover at 32 bits. The 64-bit motor feedback value is created from the 32-bit position feedback from the motor. Modulo is generally used on encoders that provide 32 bits of feedback and the user needs to keep track of the number of times a certain value is exceeded. For example, a motor with 2000 counts/rev could use a modulo value of 2000 so that the number of revolutions is counted. In this case, the upper 32 bits of the motor feedback value represents the number of revolutions and the lower 32 bits of the motor feedback value represents the position (0 – 1999) of the motor within one revolution.

	In the case where an encoder does not provide 32 bits of feedback, the reverse field is used. For example, if an encoder has 24 bits of feedback, set the modulo value to 2^{24} and set the reverse field to TRUE. The controller will create a full 64-bit encoder position based on the 24-bit feedback from the motor.

See Also

MEIMotorEncoderRatio

Definition

```
typedef struct MEIMotorEncoderRatio {
    long    A;    /* denominator */
    long    B;    /* numerator */
} MEIMotorEncoderRatio;
```

Description

MEIMotorEncoderRatio is used to set the Encoder ratio. Ratio is programmed into the firmware via a denominator (A) and a numerator (B). B/A = ratio.

NOTE: Custom Firmware is required to support this data type.

A	the denominator.
B	the numerator.

See Also

MEIMotorEncoderReverseModulo

Definition

```
typedef struct MEIMotorEncoderReverseModulo {
    long    *Ptr;   /* XMP Address */
    long    Rollover;
} MEIMotorEncoderReverseModulo;
```

Description

MEIMotorEncoderReverseModulo is used to program the firmware to calculate a 32-bit encoder position based off of the data that is pointed to by ***Ptr***. Each sample, the firmware calculates a delta from the data that is pointed to by ***Ptr*** and the ***Rollover*** value. This delta is added to a 32-bit counter to create a 32-bit position.

NOTE: Custom Firmware is required to support this data type.

***Ptr**	a pointer to the XMP address.
Rollover	the value is the resolution of the incoming data pointed to by *Ptr.

See Also

MEIMotorEncoderSsiConfig

Definition

```
typedef struct MEIMotorEncoderSsiConfig {
    MEIMotorSsiInput   input;
    long               bitCount;
    long               baudRate;  /* units = hertz */
    MPI_BOOL           brokenWireEnable;
} MEIMotorEncoderSsiConfig;
```

Required Header: stdmei.h
Change History: Added in the 03.04.00

Description

MEIMotorEncoderSsiConfig contains configurations for SSI (Synchronous-Serial Interface) feedback devices. Use this structure to configure the SSI when the MEIMotorEncoderType is configured for **MEIMotorEncoderTypeSSI**. Be sure to verify that the SynqNet node FPGA supports SSI feedback devices. See the Node FPGA Images: Features Table.

input	Source pin for the serial input from the SSI feedback device.
bitCount	Resolution in bits for the SSI feedback device. The valid range is 1 to 32 bits.
baudRate	Serial communication rate in bits per second. The valid range is 10000 to 500000.
brokenWireEnable	Enable (TRUE) or disable (FALSE) broken wire detection.

Sample Code

Configure the motor's primary feedback for an SSI device by using general purpose bit #0 to drive the clock output at 500 kHz. The serial input is decoded via the encoder input channel A.

```
returnValue =
    mpiMotorConfigGet(motor,
                      NULL,
                      &motorConfig);

 /* Configure the IO to output the SSI Clock */
   motorConfig.Io[MEIMotorIoConfigIndex0}.Type = MEIMotorIoTypeSSI_CLOCK0;

   /* Configure the primary feedback for SSI */
   motorConfig.Encoder[0].type = MEIMotorEncoderTypeSSI;
   motorConfig.Encoder[0].ssiConfig.baudRate = 500000; /* 500 KHZ */
   motorConfig.Encoder[0].ssiConfig.bitCount = 32;
   motorConfig.Encoder[0].ssiConfig.input = MEIMotorSsiInputENC_A;
   motorConfig.Encoder[0].ssiConfig.brokenWireEnable = TRUE;

returnValue =
   mpiMotorConfigSet(motor,
                     NULL,
                     &motorConfig);
```

See Also

MEIMotorSsiInput | MEIMotorEncoderType | mpiMotorConfigSet | mpiMotorConfigGet

Sample Application
ssiEncCfg.c

MEIMotorEncoderType

Definition

```
typedef enum MEIMotorEncoderType{
    MEIMotorEncoderTypeQUAD_AB = MEIXmpMotorEncoderConfigTypeQUAD_AB,
    MEIMotorEncoderTypeDRIVE   = MEIXmpMotorEncoderConfigTypeDRIVE,
    MEIMotorEncoderTypeSSI     = MEIXmpMotorEncoderConfigTypeSSI,
} MEIMotorEncoderType;
```

Required Header: stdmei.h
Change History: Modified in the 03.04.00

Description

MEIMotorEncoderType is the type of encoder being used for feedback. The encoder type is drive specific. Therefore, an appropriate default value should be set by the MPI.

MEIMotorEncoderTypeQUAD_AB	Quadrature encoder feedback.
MEIMotorEncoderTypeDRIVE	Drive memory interface feedback. This includes all feedback types supported by the drive.
MEIMotorEncoderTypeSSI	Synchronous Serial Interface (SSI) absolute encoder feedback.

See Also

MPIMotorEventConfig / MEIMotorEventConfig

Definition: MPIMotorEventConfig

```
typedef struct MPIMotorEventConfig {
    MPIAction            action;
    MPIMotorEventTrigger trigger;
    MPI_BOOL             direction;
    float                duration;   /* seconds */
} MPIMotorEventConfig;
```

Change History: Modified in the 03.03.00

Description

MPIMotorEventConfig is a structure used to configure Motor Events.

action	The action to be taken on the associated axis when the event is triggered (when the event status changes from FALSE to TRUE).
trigger	The trigger configuration for an event.
direction	Only used for Hardware and Software limits. Setting ***direction*** to TRUE requires the direction of motion to be in direction of the Limit to trigger the event. If ***direction*** is set to FALSE, a limit event can be generated regardless of the direction of motion.
duration	time that Limit (e.g. Home, Pos. and Neg. Limits, User Limit) must be asserted before Event is generated. Value in seconds. **NOTE**: The duration parameter for the home limit should be zero for standard configurations. A non-zero value will result in the home limit being missed.

Definition: MEIMotorEventConfig

```
typedef MEIXmpLimitData MEIMotorEventConfig;

typedef struct {
    MEIXmpLimitCondition    Condition[MEIXmpLimitConditions];
    MEIXmpStatus            Status;
    MEIXmpLogic             Logic;
    MEIXmpLimitOutput       Output;
    long                    Count;
    long                    State;
} MEIXmpLimitData;
```

Description

MEIMotorEventConfig is an enumeration of encoder feedback inputs for a motor.

condition	is a structure that configures the conditional statements evaluated to generate a Limit Event. Each limit may have up to two conditions (MEIXmpLimitConditions = 2). This structure is described in further detail on the User Limits page.
status	an enum that defines what actions the XMP will take when a user limit evaluates TRUE. Always set Status to at least MEIXmpStatusLIMIT to notify the motor object that a limit has occurred. Valid **Status** values are listed in the Values of Status table below.
logic	an enum that sets the logic applied between the two condition block outputs, Condition[0] and Condition[1]. Valid **Logic** values are listed in the Values of Logic table below.
output	is a structure that allows specific action to be taken when the Limit Event is generated. In addition to generating a Motion Action, a Limit can write to any other valid XMP Firmware register defined in the *OutputPtr with value described by AndMask and OrMask. This structure is described in further detail on the User Limits page.
count	For internal use only. The MPI method, mpiMotorEventConfigSet(...) will not write these values.
state	For internal use only. The MPI method, mpiMotorEventConfigSet(...) will not write these values.

Values of Status	Action to be taken
MEIXmpStatusLIMIT	None
MEIXmpStatusLIMIT | MEIXmpStatusPAUSE	Axes attached to the motor will be Paused
MEIXmpStatusLIMIT | MEIXmpStatusSTOP	Axes attached to the motor will be Stopped
MEIXmpStatusLIMIT | MEIXmpStatusABORT	Axes attached to the motor will be Aborted
MEIXmpStatusLIMIT | MEIXmpStatusESTOP	Axes attached to the motor will be E-Stopped

MEIXmpStatusLIMIT | MEIXmpStatusESTOP_ABORT	Axes attached to the motor will be E-Stopped and Aborted

Values of Logic	Evaluates	Motor object notified that a limit has occurred if...
MEIXmpLogicNEVER	Nothing	No event is generated
MEIXmpLogicSINGLE	Condition[0]	Condition[0] = = TRUE
MEIXmpLogicOR	Condition[0], Condition[1]	(Condition[0] || Condition[1]) = = TRUE
MEIXmpLogicAND	Condition[0], Condition[1]	(Condition[0] && Condition[1]) = = TRUE
other **MEIXmpLogic** enums	For internal use only.	

See Also

MPIMotorEventConfig and Motor Limit Configuration
Error Limit and Limit Switch Errors
User Limits

mpiMotorEventConfigGet | mpiMotorEventConfigSet | MPIEncoderFault

MPIMotorEventTrigger

Definition

```
typedef union {
    long                 polarity; /* 0 => active low, else active high */
    double               position; /* MPIEventTypeLIMIT_SW_[POS|NEG] */
    float                error;    /* MPIEventTypeLIMIT_ERROR */
    MPIMotorEncoderFault encoder;  /* MPIEventTypeENCODER_FAULT */
} MPIMotorEventTrigger;
```

Change History: Modified in the 03.04.00. Changed **position** info to a double.

Description

polarity	Configures the polarity for the motor event trigger. If polarity = 0 (FALSE), the event will trigger on an active low signal. If polarity = 1 (TRUE), the event will trigger on an active high signal.
position	Configures the positive and negative software position limits. The controller monitors the actual position and compares it to the positive and negative software position limits. If the positive limit is exceeded the controller will generate a MPIEventTypeLIMIT_SW_POS event. If the negative limit is exceeded the controller will generate a MPIEventTypeLIMIT_SW_NEG event. **MPI version 03.04.xx (and newer)** If the actual position value minus the negative limit value is greater than 2^{63} counts, then the negative limit will trip. Similarly, if the actual position value minus the positive limit value is less than -2^{63} counts, then the positive limit will trip. This is a result of a wraparound with the 64-bit signed value. When these conditions occur, the comparisons will not work correctly. **MPI version 03.03.xx (and older)** If the actual position value minus the negative limit value is greater than 2^{31} counts, then the negative limit will trip. Similarly, if the actual position value minus the positive limit value is less than -2^{31} counts, then the positive limit will trip. This is a result of a wraparound with the 32-bit signed value. When these conditions occur, the comparisons will not work correctly.
error	Configures the position error limit. The controller calculates the position error each sample period: (command position - actual position) = position error. If the position error exceeds the range of the specified error limit, the controller will generate a MPIEventTypeLIMIT_ERROR.

encoder	Configures the controller to monitor the primary or secondary for faults. The encoder should be set to one of the values defined by the MPIMotorEncoderFault {...} enumeration. When configured, if the primary, secondary, or either encoder generates a fault, the controller will generate the MPIEventTypeENCODER_FAULT.

See Also

MPIMotorEventConfig | MPIMotorEncoderFault | MPIEventType | MEIEventType | Error Limit and Limit Switch Errors

MEIMotorFaultBit

Definition

```
typedef enum MEIMotorFaultBit {
    MEIMotorFaultBitINVALID,
    MEIMotorFaultBitAMP_FAULT,
    MEIMotorFaultBitDRIVE_FAULT,
    MEIMotorFaultBitWATCHDOG_FAULT,
    MEIMotorFaultBitCHECKSUM_ERROR,
    MEIMotorFaultBitFEEDBACK_FAULT,
    MEIMotorFaultBitAMP_NOT_POWERED,
    MEIMotorFaultBitDRIVE_NOT_READY,
    MEIMotorFaultBitFEEDBACK_FAULT_SECONDARY,
} MEIMotorFaultBit;
```

Description

MEIMotorFaultBit is an enumeration of motor fault bits for the SynqNet node drive interface. The support for motor fault bits is node/drive specific. Please see the node\drive manufacturer's documentation for details.

MEIMotorFaultBitAMP_FAULT

External amp fault input pin. Indicates a fault condition in the amplifier.

Available on remote motion blocks.

MEIMotorFaultBitDRIVE_FAULT

Bit in the drive interface status register. Indicates a fault condition in the drive.

NOT available on remote motion blocks.

MEIMotorFaultBitWATCHDOG_FAULT

Bit in the drive interface status register. Indicates a drive failure due to a watchdog timeout. The node alternately sets/clears the watchdog each sample, the drive's processor then clears/sets the watchdog. If the drive's processor does not respond, the watchdog fault bit is set.

NOT available on remote motion blocks.

MEIMotorFaultBitCHECKSUM_ERROR

Bit in the drive interface status register. Indicates the data transfer (via serial or parallel memory interface) between the failed SynqNet node FPGA and drive with a checksum error.

NOT available on remote motion blocks.

MEIMotorFaultBitFEEDBACK_FAULT

Bit in the drive interface status register. Active when one or more of the feedback status bits is triggered. Indicates that the drive/motor position feedback system has failed.

Available on remote motion blocks.

MEIMotorFaultBitAMP_NOT_POWERED

Bit in the drive interface status register. The SynqNet drive amplifier power stage does not have sufficient voltage. The motor windings cannot be energized properly until this bit is clear.

NOT available on remote motion blocks.

MEIMotorFaultBitDRIVE_NOT_READY

Bit in the drive interface status register. The SynqNet drive is not ready to receive commands or servo motors. This fault indicates the drive has not completed its processor initialization or there is some other serious drive processor problem.

NOT available on remote motion blocks.

MEIMotorFaultBitFEEDBACK_FAULT_SECONDARY

Bit in the drive interface status register. Active when one or more of the secondary feedback status bits is triggered. Indicates that the drive/motor auxilury position feedback system has failed.

Available on remote motion blocks.

See Also

MEIMotorFaultMask | MEIMotorFaultConfig | meiMotorStatus

MEIMotorFaultConfig

Definition

```
typedef struct MEIMotorFaultConfig {
    MEIMotorFaultMask   faultMask; /* sets the
                                      MEIMotorDedicatedInAMP_FAULT state */
} MEIMotorFaultConfig;
```

Description

MEIMotorFaultConfig specifies the motor fault bit configuration.

faultMask†	A mask of motor fault bits. The masked motor fault bits will be monitored by the controller's motor object. If any masked motor fault bit is active (TRUE), the motor's dedicated amp fault input (MEIMotorDedicatedInAMP_FAULT) is set active (TRUE). The support for motor fault bits is node/drive specific. During SynqNet network initialization, the SqNodeLib automatically configures the faultMask based on the node type. See the node/drive manufacturer's documentation for details.

† - If you attempt to change this value and hardware is not connected, an error message will be returned.

See Also

MEIMotorFaultBit | MEIMotorFaultMask | meiMotorStatus

MEIMotorFaultMask

Definition

```
typedef enum MEIMotorFaultMask {
      MEIMotorFaultMaskAMP             = (1<<
MEIMotorFaultBitAMP_FAULT),
      MEIMotorFaultMaskDRIVE           = (1<< MEIMotorFaultBitDRIVE_FAULT),
      MEIMotorFaultMaskWATCHDOG        = (1<< MEIMotorFaultBitWATCHDOG_FAULT),
      MEIMotorFaultMaskCHECKSUM        = (1<< MEIMotorFaultBitCHECKSUM_ERROR),
      MEIMotorFaultMaskFEEDBACK        = (1<< MEIMotorFaultBitFEEDBACK_FAULT),
      MEIMotorFaultMaskAMP_NOT_POWERED = (1<< MEIMotorFaultBitAMP_NOT_POWERED),
      MEIMotorFaultMaskDRIVE_NOT_READY = (1<< MEIMotorFaultBitDRIVE_NOT_READY),
      MEIMotorFaultMaskFEEDBACK_FAULT_SECONDARY = (1<<
MEIMotorFaultBitFEEDBACK_FAULT_SECONDARY),
  } MEIMotorFaultMask;
```

Description

MEIMotorFaultMask is an enumeration of motor fault bit masks for the SynqNet node drive interface. The support for motor fault bits is node/drive specific. Please see the node\drive manufacturer's documentation for details.

MEIMotorFaultMaskAMP	External amp fault input pin. Indicates a fault condition in the amplifier. Available on remote motion blocks.
MEIMotorFaultMaskDRIVE	Bit in the drive interface status register. Indicates a fault condition in the drive. NOT available on remote motion blocks.
MEIMotorFaultMaskWATCHDOG	Bit in the drive interface status register. Indicates a drive failure due to a watchdog timeout. The node alternately sets/clears the watchdog each sample, the drive's processor then clears/sets the watchdog. If the drive's processor does not respond, the watchdog fault bit is set. NOT available on remote motion blocks.
MEIMotorFaultMaskCHECKSUM	Bit in the drive interface status register. Indicates the data transfer (via serial or parallel memory interface) between the failed SynqNet node FPGA and drive with a checksum error. NOT available on remote motion blocks.

MEIMotorFaultMaskFEEDBACK	Bit in the drive interface status register. Active when one or more of the feedback status bits is triggered. Indicates that the drive/motor position feedback system has failed. Available on remote motion blocks.
MEIMotorFaultMaskAMP_NOT_POWERED	Bit in the drive interface status register. The SynqNet drive amplifier power stage does not have sufficient voltage. The motor windings cannot be energized properly until this bit is clear. NOT available on remote motion blocks.
MEIMotorFaultMaskDRIVE_NOT_READY	Bit in the drive interface status register. The SynqNet drive is not ready to receive commands or servo motors. This fault indicates the drive has not completed its processor initialization or there is some other serious drive processor problem. NOT available on remote motion blocks.
MEIMotorFaultMaskFEEDBACK_FAULT_SECONDARY	Bit in the drive interface status register. Active when one or more of the secondary feedback status bits is triggered. Indicates that the drive/motor auxilury position feedback system has failed. Available on remote motion blocks.

See Also

MEIMotorFaultBit | MEIMotorFaultConfig | meiMotorStatus

MPIMotorFeedback

Definition

```
typedef   double   MPIMotorFeedback[MPIMotorEncoderLAST];
```

Description

MPIMotorFeedback is an array of doubles used to store motor feedback (both primary and secondary).

See Also

MPIMotorEncoder | mpiMotorFeedback

MPIMotorGeneralIo

Definition

```
typedef enum MPIMotorGeneralIo {
    MPIMotorGeneralIo0,
    MPIMotorGeneralIo1,
    MPIMotorGeneralIo2,
    MPIMotorGeneralIo3,
    MPIMotorGeneralIo4,
    MPIMotorGeneralIo5,
    MPIMotorGeneralIo6,
    MPIMotorGeneralIo7,
    MPIMotorGeneralIo8,
    MPIMotorGeneralIo9,
    MPIMotorGeneralIo10,
    MPIMotorGeneralIo11,
    MPIMotorGeneralIo12,
    MPIMotorGeneralIo13,
    MPIMotorGeneralIo14,
    MPIMotorGeneralIo15,
    MPIMotorGeneralIo16,
    MPIMotorGeneralIo17,
    MPIMotorGeneralIo18,
    MPIMotorGeneralIo19,
    MPIMotorGeneralIo20,
    MPIMotorGeneralIo21,
    MPIMotorGeneralIo22,
    MPIMotorGeneralIo23,
    MPIMotorGeneralIo24,
    MPIMotorGeneralIo25,
    MPIMotorGeneralIo26,
    MPIMotorGeneralIo27,
    MPIMotorGeneralIo28,
    MPIMotorGeneralIo29,
    MPIMotorGeneralIo30,
    MPIMotorGeneralIo31,
} MPIMotorGeneralIo;
```

Change History: Modified in the 03.04.00. Added in the 03.02.00.
MPIMotorGeneralIo replaced MEIMotorIoMask.

Description

MPIMotorGeneralIo enumeration gives labels for each of the general purpose outputs that a motor can support.

NOTE: MPIMotorGeneralIo replaced MEIMotorIoMask in the MPI library.

MPIMotorGeneralIo0	General I/O bit 0.
MPIMotorGeneralIo1	General I/O bit 1.
MPIMotorGeneralIo2	General I/O bit 2.
MPIMotorGeneralIo3	General I/O bit 3.
MPIMotorGeneralIo4	General I/O bit 4.
MPIMotorGeneralIo5	General I/O bit 5.
MPIMotorGeneralIo6	General I/O bit 6.
MPIMotorGeneralIo7	General I/O bit 7.
MPIMotorGeneralIo8	General I/O bit 8.
MPIMotorGeneralIo9	General I/O bit 9.
MPIMotorGeneralIo10	General I/O bit 10.
MPIMotorGeneralIo11	General I/O bit 11.
MPIMotorGeneralIo12	General I/O bit 12.
MPIMotorGeneralIo13	General I/O bit 13.
MPIMotorGeneralIo14	General I/O bit 14.
MPIMotorGeneralIo15	General I/O bit 15.
MPIMotorGeneralIo16	General I/O bit 16.
MPIMotorGeneralIo17	General I/O bit 17.
MPIMotorGeneralIo18	General I/O bit 18.
MPIMotorGeneralIo19	General I/O bit 19.
MPIMotorGeneralIo20	General I/O bit 20.
MPIMotorGeneralIo21	General I/O bit 21.
MPIMotorGeneralIo22	General I/O bit 22.
MPIMotorGeneralIo23	General I/O bit 23.
MPIMotorGeneralIo24	General I/O bit 24.
MPIMotorGeneralIo25	General I/O bit 25.

MPIMotorGeneralIo26	General I/O bit 26.
MPIMotorGeneralIo27	General I/O bit 27.
MPIMotorGeneralIo28	General I/O bit 28.
MPIMotorGeneralIo29	General I/O bit 29.
MPIMotorGeneralIo30	General I/O bit 30.
MPIMotorGeneralIo31	General I/O bit 31.

See Also

Please see General Purpose I/O

MEIMotorInfo

Definition

```
typedef struct MEIMotorInfo {
    MEIMotorInfoNodeType   nodeType;
    struct {
        long    networkNumber;
        long    nodeNumber;
        long    driveIndex;
    } sqNode;

    long    captureCount;
    long    encoderCount;
    long    probeCount;

    MEIMotorInfoDedicatedIn    dedicatedIn[MPIMotorDedicatedInLAST];
    MEIMotorInfoDedicatedOut   dedicatedOut[MPIMotorDedicatedOutLAST];
    MEIMotorInfoGeneralIo      generalIo[MPIMotorGeneralIoLAST];
} MEIMotorInfo;
```

Change History: Modified in the 03.02.00

Description

MEIMotorInfo structure contains static data determined during network initialization. It identifies the network, node, and drive interface associated with the motor object.

nodeType	Identifies the type of node. See MEIMotorInfoNodeType.
networkNumber	An index to the SynqNet network (0, 1, 2 etc.).
nodeNumber	An index to the node (0, 1, 2, etc.).
driveIndex	An index to the drive interface (0, 1, 2, etc.).
captureCount	Counts the number of captures for the motor.
encoderCount	Counts the number of encoders for the motor.
probeCount	Counts the number of hardware Probe engines for the motor.
dedicatedIn	Details about the dedicated inputs.
dedicatedOut	Details about the dedicated outputs.

generalIo	Details about the general purpose I/O.

See Also

meiMotorInfo

MEIMotorInfoDedicatedIn

Definition

```
typedef struct MEIMotorInfoDedicatedIn {
   MPI_BOOL    supported;
} MEIMotorInfoDedicatedIn;
```

Change History: Changed in the 03.03.00. Added in the 03.02.00.

Description

supported	0 = This dedicated input is NOT supported by the hardware. 1 = This dedicated input is supported by the hardware.

See Also

MEIMotorInfoDedicatedOut

MEIMotorInfoDedicatedOut

Definition

```
typedef struct MEIMotorInfoDedicatedOut {
  MPI_BOOL   supported;
} MEIMotorInfoDedicatedOut;
```

Change History: Modified in the 03.03.00. Added in the 03.02.00

Description

supported	0 = This dedicated output is NOT supported by the hardware. 1 = This dedicated output is supported by the hardware.

See Also

MEIMotorInfoDedicatedIn

MEIMotorInfoGeneralIo

Definition

```
typedef struct MEIMotorInfoGeneralIo {
   MPI_BOOL             supported;
   char                 *name;
   MEIMotorIoTypeMask   validTypes;
} MEIMotorInfoGeneralIo;
```

Change History: Modified in the 03.03.00. Added in the 03.02.00.

Description

supported	0 = This dedicated output is NOT supported by the hardware. 1 = This dedicated output is supported by the hardware.
***name**	The name the node uses for this I/O pin
validTypes	This is a bit field, each bit indicates a valid source/input that the node supports.

See Also

MEIMotorInfoNodeType

Definition

```
typedef enum MEIMotorInfoNodeType {
    MEIMotorInfoNodeTypeNONE,
    MEIMotorInfoNodeTypeSQNODE,
} MEIMotorInfoNodeType;
```

Description

MEIMotorInfoNodeType is an enumeration of node types. It specifies the node type associated with the motor.

MEIMotorInfoNodeTypeNONE	Not a network node. The node is local to the controller.
MEIMotorInfoNodeTypeSQNODE	A SynqNet node type.

See Also

MEIMotorInfo

MEIMotorIoConfig

Definition

```
typedef struct MEIMotorIoConfig {
    MEIMotorIoType     Type;
} MEIMotorIoConfig;
```

Description

MEIMotorIoConfig specifies the configuration for the motor's digital I/O.

Type	The I/O bit function. See MEIMotorIoType. If you attempt to change this value and hardware is not connected, an error message will be returned.

See Also

mpiMotorConfigGet | mpiMotorConfigSet

MEIMotorIoConfigIndex

Definition

```
typedef enum MEIMotorIoConfigIndex {
    MEIMotorIoConfigIndex0,
    MEIMotorIoConfigIndex1,
    MEIMotorIoConfigIndex2,
    MEIMotorIoConfigIndex3,
    MEIMotorIoConfigIndex4,
    MEIMotorIoConfigIndex5,
    MEIMotorIoConfigIndex6,
    MEIMotorIoConfigIndex7,
    MEIMotorIoConfigIndex8,
    MEIMotorIoConfigIndex9,
    MEIMotorIoConfigIndex10,
    MEIMotorIoConfigIndex11,
    MEIMotorIoConfigIndex12,
    MEIMotorIoConfigIndex13,
    MEIMotorIoConfigIndex14,
    MEIMotorIoConfigIndex15,
    MEIMotorIoConfigIndex16,
    MEIMotorIoConfigIndex17,
    MEIMotorIoConfigIndex18,
    MEIMotorIoConfigIndex19,
    MEIMotorIoConfigIndex20,
    MEIMotorIoConfigIndex21,
    MEIMotorIoConfigIndex22,
    MEIMotorIoConfigIndex23,
    MEIMotorIoConfigIndex24,
    MEIMotorIoConfigIndex25,
    MEIMotorIoConfigIndex26,
    MEIMotorIoConfigIndex27,
    MEIMotorIoConfigIndex28,
    MEIMotorIoConfigIndex29,
    MEIMotorIoConfigIndex30,
    MEIMotorIoConfigIndex31,

} MEIMotorIoConfigIndex;
```

Change History: Modified in the 03.04.00.

Description

MEIMotorIoConfigIndex is an enumeration of configurable motor I/O, indexed by a number.

MEIMotorIoConfigIndex0	motor I/O number 0
MEIMotorIoConfigIndex1	motor I/O number 1
MEIMotorIoConfigIndex2	motor I/O number 2
MEIMotorIoConfigIndex3	motor I/O number 3
MEIMotorIoConfigIndex4	motor I/O number 4
MEIMotorIoConfigIndex5	motor I/O number 5
MEIMotorIoConfigIndex6	motor I/O number 6
MEIMotorIoConfigIndex7	motor I/O number 7
MEIMotorIoConfigIndex8	motor I/O number 8
MEIMotorIoConfigIndex9	motor I/O number 9
MEIMotorIoConfigIndex10	motor I/O number 10
MEIMotorIoConfigIndex11	motor I/O number 11
MEIMotorIoConfigIndex12	motor I/O number 12
MEIMotorIoConfigIndex13	motor I/O number 13
MEIMotorIoConfigIndex14	motor I/O number 14
MEIMotorIoConfigIndex15	motor I/O number 15
MEIMotorIoConfigIndex16	motor I/O number 16
MEIMotorIoConfigIndex17	motor I/O number 17
MEIMotorIoConfigIndex18	motor I/O number 18
MEIMotorIoConfigIndex19	motor I/O number 19
MEIMotorIoConfigIndex20	motor I/O number 20
MEIMotorIoConfigIndex21	motor I/O number 21
MEIMotorIoConfigIndex22	motor I/O number 22
MEIMotorIoConfigIndex23	motor I/O number 23
MEIMotorIoConfigIndex24	motor I/O number 24
MEIMotorIoConfigIndex25	motor I/O number 25
MEIMotorIoConfigIndex26	motor I/O number 26

MEIMotorIoConfigIndex27	motor I/O number 27
MEIMotorIoConfigIndex28	motor I/O number 28
MEIMotorIoConfigIndex29	motor I/O number 29
MEIMotorIoConfigIndex30	motor I/O number 30
MEIMotorIoConfigIndex31	motor I/O number 31

See Also

MEIMotorIoConfig

MEIMotorIoType

Definition

```
typedef enum MEIMotorIoType {
   MEIMotorIoTypeOUTPUT,

   MEIMotorIoTypePULSE_A,
   MEIMotorIoTypePULSE_B,
   MEIMotorIoTypeCOMPARE_0,
   MEIMotorIoTypeCOMPARE_1,
   MEIMotorIoTypeSOURCE5
   MEIMotorIoTypeBRAKE,

   MEIMotorIoTypeSSI_CLOCK0,
   MEIMotorIoTypeSSI_CLOCK1,

   MEIMotorIoTypeSOURCE9,
   MEIMotorIoTypeSOURCE10,
   MEIMotorIoTypeSOURCE11,
   MEIMotorIoTypeSOURCE12,
   MEIMotorIoTypeSOURCE13,
   MEIMotorIoTypeSOURCE14,
   MEIMotorIoTypeSOURCE15,

   MEIMotorIoTypeINPUT,

   MEIMotorIoTypeSOURCE1 = MEIMotorIoTypePULSE_A,
   MEIMotorIoTypeSOURCE2 = MEIMotorIoTypePULSE_B,
   MEIMotorIoTypeSOURCE3 = MEIMotorIoTypeCOMPARE_0,
   MEIMotorIoTypeSOURCE4 = MEIMotorIoTypeCOMPARE_1,

   MEIMotorIoTypeSOURCE6 = MEIMotorIoTypeBRAKE,
   MEIMotorIoTypeSOURCE7 = MEIMotorIoTypeSSI_CLOCK0,
   MEIMotorIoTypeSOURCE8 = MEIMotorIoTypeSSI_CLOCK1,
} MEIMotorIoType;
```

Change History: Modified in the 03.04.00. Modified in the 03.02.00.

Description

MEIMotorIoType is an enumeration of motor I/O functions. A SynqNet node's motor I/O may support one or more features depending on the node hardware and FPGA. MotorIoType contains the generic

enumerations for all nodes. For the node specific enumerations, please see the individual SqNode modules for details.

MEIMotorIoTypeOUTPUT	discrete digital output
MEIMotorIoTypeSOURCE1	motor I/O source number 1 for node-specific feature.
MEIMotorIoTypeSOURCE2	motor I/O source number 2 for node-specific feature.
MEIMotorIoTypeSOURCE3	motor I/O source number 3 for node-specific feature.
MEIMotorIoTypeSOURCE4	motor I/O source number 4 for node-specific feature.
MEIMotorIoTypeSOURCE5	motor I/O source number 5 for node-specific feature.
MEIMotorIoTypeSOURCE6	motor I/O source number 6 for node-specific feature.
MEIMotorIoTypeSOURCE7	motor I/O source number 7 for node-specific feature.
MEIMotorIoTypeSOURCE8	motor I/O source number 8 for node-specific feature.
MEIMotorIoTypeSOURCE9	motor I/O source number 9 for node-specific feature.
MEIMotorIoTypeSOURCE10	motor I/O source number 10 for node-specific feature.
MEIMotorIoTypeSOURCE11	motor I/O source number 11 for node-specific feature.
MEIMotorIoTypeSOURCE12	motor I/O source number 12 for node-specific feature.
MEIMotorIoTypeSOURCE13	motor I/O source number 13 for node-specific feature.
MEIMotorIoTypeSOURCE14	motor I/O source number 14 for node-specific feature.
MEIMotorIoTypeSOURCE15	motor I/O source number 15 for node-specific feature.
MEIMotorIoTypeINPUT	discrete digital input

See Also

mpiMotorConfigGet | mpiMotorConfigSet

MEIMotorIoTypeMask

Definition

```
typedef enum MEIMotorIoTypeMask {
  MEIMotorIoTypeMaskOUTPUT   = (1<<MEIMotorIoTypeOUTPUT),
  MEIMotorIoTypeMaskSOURCE1  = (1<<MEIMotorIoTypeSOURCE1),
  MEIMotorIoTypeMaskSOURCE2  = (1<<MEIMotorIoTypeSOURCE2),
  MEIMotorIoTypeMaskSOURCE3  = (1<<MEIMotorIoTypeSOURCE3),
  MEIMotorIoTypeMaskSOURCE4  = (1<<MEIMotorIoTypeSOURCE4),
  MEIMotorIoTypeMaskSOURCE5  = (1<<MEIMotorIoTypeSOURCE5),
  MEIMotorIoTypeMaskSOURCE6  = (1<<MEIMotorIoTypeSOURCE6),
  MEIMotorIoTypeMaskSOURCE7  = (1<<MEIMotorIoTypeSOURCE7),
  MEIMotorIoTypeMaskSOURCE8  = (1<<MEIMotorIoTypeSOURCE8),
  MEIMotorIoTypeMaskSOURCE9  = (1<<MEIMotorIoTypeSOURCE9),
  MEIMotorIoTypeMaskSOURCE10 = (1<<MEIMotorIoTypeSOURCE10),
  MEIMotorIoTypeMaskSOURCE11 = (1<<MEIMotorIoTypeSOURCE11),
  MEIMotorIoTypeMaskSOURCE12 = (1<<MEIMotorIoTypeSOURCE12),
  MEIMotorIoTypeMaskSOURCE13 = (1<<MEIMotorIoTypeSOURCE13),
  MEIMotorIoTypeMaskSOURCE14 = (1<<MEIMotorIoTypeSOURCE14),
  MEIMotorIoTypeMaskSOURCE15 = (1<<MEIMotorIoTypeSOURCE15),

  MEIMotorIoTypeMaskINPUT = (1<<MEIMotorIoTypeINPUT),

  MEIMotorIoTypeMaskPULSE_A = (1<<MEIMotorIoTypePULSE_A),
  MEIMotorIoTypeMaskPULSE_B = (1<<MEIMotorIoTypePULSE_B),

  MEIMotorIoTypeMaskCOMPARE_0 = (1<<MEIMotorIoTypeCOMPARE_0),
  MEIMotorIoTypeMaskCOMPARE_1 = (1<<MEIMotorIoTypeCOMPARE_1),

  MEIMotorIoTypeMaskBRAKE = (1<<MEIMotorIoTypeBRAKE),

  MEIMotorIoTypeMaskSSI_CLOCK0 = (1<<MEIMotorIoTypeSSI_CLOCK0),
  MEIMotorIoTypeMaskSSI_CLOCK1 = (1<<MEIMotorIoTypeSSI_CLOCK1),
} MEIMotorIoTypeMask;
```

Change History: Added in the 03.02.00

Description

MEIMotorIoTypeMask is an enumeration to mask bits from the MEIMotorInfo structure.

MEIMotorIoTypeMaskOUTPUT	Supports the output configuration.
MEIMotorIoTypeMaskPULSE_A	Supports the Pulse A configuration.
MEIMotorIoTypeMaskPULSE_B	Supports the Pulse B configuration.
MEIMotorIoTypeMaskCOMPARE_0	Currently not supported.
MEIMotorIoTypeMaskCOMPARE_1	Currently not supported.
MEIMotorIoTypeMaskBRAKE	See MPIMotorBrake
MEIMotorIoTypeMaskSSI_CLOCK0	Supports the SSI_CLOCK0 configuration.
MEIMotorIoTypeMaskSSI_CLOCK1	Supports the SSI_CLOCK1 configuration.
MEIMotorIoTypeMaskINPUT	Supports the input configuration.

See Also

MPIMotorBrake | MEIMotorInfo | mpiMotorConfigGet | mpiMotorConfigSet | MEIMotorIoType

MPIMotorMessage / MEIMotorMessage

Definition: MPIMotorMessage

```
typedef enum {
     MPIMotorMessageMOTOR_INVALID,
     MPIMotorMessageTYPE_INVALID,
     MPIMotorMessageSTEPPER_NA,
} MPIMotorMessage;
```

Required Header: stdmpi.h
Change History: Modified in the 03.04.00: Addition of STEPPER_NA. Modified in the 03.02.00.

Description

MPIMotorMessage is an enumeration of Motor error messages that can be returned by the MPI library.

MPIMotorMessageMOTOR_INVALID

The motor number is out of range. This message code is returned by mpiMotorCreate(…) if the motor number is less than zero or greater than or equal to MEIXmpMAX_Motors.

MPIMotorMessageTYPE_INVALID

The motor type is not valid. This message code is returned by mpiMotorConfigGet / Set(...) if the motor type is not a value defined in the enumeration MPIMotorType. To avoid this error, use one of the defined motor types in MPIMotorType.

MPIMotorMessageSTEPPER_NA

The hardware does not support the Stepper motor type. This message code is returned by mpiMotorConfigSet(...) if the motor type is configured for MPIMotorTypeSTEPPER when the node hardware does not support steppers. To correct the problem, do not select the stepper motor type.

Definition: MEIMotorMessage

```
typedef enum {
    MEIMotorMessageMOTOR_NOT_ENABLED,
    MEIMotorMessageSECONDARY_ENCODER_NA,
    MEIMotorMessageHARDWARE_NOT_FOUND,
    MEIMotorMessageSTEPPER_INVALID,
    MEIMotorMessageDISABLE_ACTION_INVALID,
    MEIMotorMessagePULSE_WIDTH_INVALID,
    MEIMotorMessageFEEDBACK_REVERSAL_NA,
    MEIMotorMessageFILTER_DISABLE_NA,
    MEIMotorMessagePHASE_FINDING_FAILED,
    MEIMotorMessageDEMAND_MODE_UNSUPPORTED,
    MEIMotorMessageDEMAND_MODE_NOT_SET,
} MEIMotorMessage;
```

Required Header: stdmei.h
Change History: Modified in the 03.04.00. Modified in the 03.03.00.

Description

MEIMotorMessage is an enumeration of Motor error messages that can be returned by the MPI library.

MEIMotorMessageMOTOR_NOT_ENABLED

The motor number is not active in the controller. This message code is returned by mpiMotorEventConfigGet(…) if the specified motor is not enabled in the controller. To correct the problem, use mpiControlConfigSet(…) to enable the motor object, by setting the motorCount to greater than the motor number. For example, to enable motor 0 to 3, set motorCount to 4.

MEIMotorMessageSECONDARY_ENCODER_NA

The motor's secondary encoder is not available. This message code is returned by mpiMotorConfigSet(…) or mpiMotorEventConfigSet(…) if the encoder fault trigger is configured for a secondary encoder when the hardware does not support a secondary encoder. To correct the problem, do not select the secondary encoder when configuring the encoder fault conditions.

MEIMotorMessageHARDWARE_NOT_FOUND

The motor object's hardware resource is not available. This message code is returned by mpiMotorConfigGet(…) or mpiMotorConfigSet(…) if the node hardware for the motor is not found. During controller and network initialization the nodes and motor count for each node is discovered and mapped to the controller's motor objects. An application should not configure a motor object if there is no mapped hardware to receive the service commands. To correct this problem, verify that all expected nodes were found. Use meiSqNodeInfo(…) to determine the node topology and motor count per node. Check the node hardware power and network connections.

MEIMotorMessageSTEPPER_INVALID

The motor object stepper configuration is not valid. These message codes are returned by mpiMotorConfigGet(…) or mpiMotorConfigSet(…) if the motor type is configured for stepper while the disable action is configured for command position equals actual position. The disable action feature sets the command position equal to the actual position when the amp enable signal is set to disable. Stepper motor types are driven by a digital pulse, which is triggered by the controller's command position. Do not use disable action set to command equals actual with stepper motor types.

MEIMotorMessageDISABLE_ACTION_INVALID

The motor object stepper configuration is not valid. These message codes are returned by mpiMotorConfigGet(…) or mpiMotorConfigSet(…) if the motor type is configured for stepper while the disable action is configured for command position equals actual position. The disable action feature sets the command position equal to the actual position when the amp enable signal is set to disable. Stepper motor types are driven by a digital pulse, which is triggered by the controller's command position. Do not use disable action set to command equals actual with stepper motor types.

MEIMotorMessagePULSE_WIDTH_INVALID

The motor stepper pulse width is not valid. This message code is returned by mpiMotorConfigSet(...) if the pulseWidth is out of range. To correct the problem, specify the the pulse width value between 100 nanoseconds and 1 millisecond.

MEIMotorMessageFEEDBACK_REVERSAL_NA

The feedback reversal is not applicable for the feedback type specified. This message code is returned by mpiMotorConfigSet(...) if the feedback type is not MEIMotorEncoderTypeQUAD_AB and the encoderPhase is set to TRUE. To correct the problem, set encoderPhase to FALSE. Some drives may support feedback reversal via drive parameters. Please consult the drive manufacturer's documentation for details.

MEIMotorMessageFILTER_DISABLE_NA

The feedback filter disable is not applicable for the feedback type specified. This message code is returned by mpiMotorConfigSet(...) if the feedback type is not MEIMotorEncoderTypeQUAD_AB and the filterDisable is set to TRUE. To correct the problem, set filterDisable to FALSE.

MEIMotorMessagePHASE_FINDING_FAILED

The drive failed to complete the phase finding procedure. This message code is returned by meiMotorPhaseFindStart(…) if the drive failed to successfully initailized commutation. To determine the cause of the failure, check the fault and warning codes from the drive with meiMotorAmpFault(…) and meiMotorAmpWarning(…).

MEIMotorMessageDEMAND_MODE_UNSUPPORTED

The motor does not support the specified demand mode. This message is returned by mpiMotorConfigSet(…) or mpiMotorEventConfigSet(…) if the demand mode is configured for a mode that the node/drive does not support. To correct this problem, use the default demand mode or specify a different demand mode.

MEIMotorMessageDEMAND_MODE_NOT_SET

The demand mode cannot be changed while the amplifier is enabled. This message is returned by mpiMotorConfigSet(…) or mpiMotorEventConfigSet(…) if the demand mode is changed while the motor's amplifier is enabled. To avoid this error message, disable the motor's amp enable with mpiMotorAmpEnableSet(…) before changing the demand mode.

See Also

mpiMotorCreate | MEIMotorEncoderType

MEIMotorPhaseFindDriveMsg

Definition

```
typedef struct MEIMotorPhaseFindDriveMsg {
/* drive specific code and message reporting information
   on the success or failure, see drive module for details */
   long   code;
   char*  message;
} MEIMotorPhaseFindDriveMsg;
```

Change History: Added in the 03.03.00

Description

MEIMotorPhaseFindDriveMsg drive specific information which reflects the quality or failure status of the phase finding routine.

NOTE: Not all drives will support this field.

code	A value retrieved from the drive which reports the status or quality of the Phase Finding Procedure. This value is drive manufacturer specific and is used for informational purposes only and is not evaluated by the controller.
message	A string provided by the drive manufacturer that describes what “code” represents.

See Also

MEIMotorPhaseFindState | MEIMotorPhaseFindStatus | meiMotorPhaseFindStart | meiMotorPhaseFindAbort | meiMotorPhaseFindStatus

Motor Phase Finding

MEIMotorPhaseFindState

Definition

```
typedef enum MEIMotorPhaseFindState  {
   MEIMotorPhaseFindStateIN_PROGRESS,
   MEIMotorPhaseFindStateSUCCESS,
   MEIMotorPhaseFindStateFAILURE,
}MEIMotorPhaseFindState;
```

Change History: Added in the 03.03.00

Description

MEIMotorPhaseFindState is used to report the state of the phase finding procedure.

MEIMotorPhaseFindStateIN_PROGRESS	The drive is still in the process of phase finding.
MEIMotorPhaseFindStateSUCCESS	The drive has successfully completed phase finding.
MEIMotorPhaseFindStateFAILURE	The drive has finished phase finding but the process was a failure.

See Also

MEIMotorPhaseFindStatus | MEIMotorPhaseFindDriveMessage | meiMotorPhaseFindStart | meiMotorPhaseFindAbort | meiMotorPhaseFindStatus

Motor Phase Finding

MEIMotorPhaseFindStatus

Definition

```
typedef struct  MEIMotorPhaseFindStatus {
   MEIMotorPhaseFindState       state;
   MEIMotorPhaseFindDriveMsg    msg;
}MEIMotorPhaseFindStatus;
```

Change History: Added in the 03.03.00

Description

MEIMotorPhaseFindStatus is a structure containing information which reflects the status of the drive Phase Finding Procedure.

state	The current state of the phase finding process: in progress, failed, success.
msg	A drive-specific code and string that describes the quality of the phase finding after the process has been completed. Or it may also include status information that describes why the phase finding failed. **NOTE**: The support for this variable is drive dependant and in some cases may not be available. Be sure to check with the drive manufacturer for support.

See Also

MEIMotorPhaseFindState | MEIMotorPhaseFindDriveMessage | meiMotorPhaseFindStart | meiMotorPhaseFindAbort | meiMotorPhaseFindStatus

Motor Phase Finding

MEIMotorSsiInput

Definition

```
typedef enum MEIMotorSsiInput{
    MEIMotorSsiInputENC_A,
    MEIMotorSsiInputENC_B,
    MEIMotorSsiInputENC_I,
    MEIMotorSsiInputGPIO_0,
    MEIMotorSsiInputGPIO_1,
    MEIMotorSsiInputGPIO_2,
    MEIMotorSsiInputGPIO_3,
    MEIMotorSsiInputGPIO_4,
    MEIMotorSsiInputGPIO_5,
    MEIMotorSsiInputGPIO_6,
} MEIMotorSsiInput;
```

Required Header: stdmei.h
Change History: Added in the 03.04.00

Description

MEIMotorSsiInput is an enumeration of input sources for SSI (Synchronous-Serial Interface) feedback devices. This enumeration is used to specify the input pin for the serial data when using the MEIMotorEncoderSsiConfig configuration structure.

MEIMotorSsiInputENC_A	Encoder input channel A
MEIMotorSsiInputENC_B	Encoder input channel B
MEIMotorSsiInputENC_I	Encoder input channel I
MEIMotorSsiInputGPIO_0	General purpose input bit 0
MEIMotorSsiInputGPIO_1	General purpose input bit 1
MEIMotorSsiInputGPIO_2	General purpose input bit 2
MEIMotorSsiInputGPIO_3	General purpose input bit 3
MEIMotorSsiInputGPIO_4	General purpose input bit 4
MEIMotorSsiInputGPIO_5	General purpose input bit 5
MEIMotorSsiInputGPIO_6	General purpose input bit 6

See Also

MEIMotorEncoderSsiConfig | mpiMotorConfigGet | mpiMotorConfigSet

MEIMotorStatus

Definition

```
typedef struct MEIMotorStatus {
    MEIMotorDacStatus       dac;
    MEIMotorFaultMask       faultMask;
    MEIMotorStepperStatus   stepper;
} MEIMotorStatus;
```

Description

MEIMotorStatus returns the specific Motor Status registers of the controller.

dac	The status for both Command and Auxiliary DACs.
faultMask	The motor fault bit masks for the SynqNet node drive interface.
stepper	Shows the status of the pulse module. It is used for checking network synchronization with the pulse module. It can also be used to check if a pulse velocity/width error occurred.

See Also

MEIMotorStepperStatus

MEIMotorStatusOutput

Definition

```
typedef struct MEIMotorStatusOutput {
    long           *outPtr;
    MEIXmpIOMask   ioMask[MEIXmpMotorStatusOutputs];
} MEIMotorStatusOutput;
```

Description

MEIMotorStatusOutput specifies which motor outputs to follow the motor status bits. This feature requires custom controller firmware.

***outPtr**	a pointer to controller memory.
ioMask	a bit mask to select the motor output signals.

See Also

mpiMotorConfigGet | mpiMotorConfigSet

MEIMotorStepper

Definition

```
typedef struct MEIMotorStepper {
    MPI_BOOL             loopback;   /* TRUE = FPGA pulse feedback,
                                        FALSE = encoder feedback */
    MEIMotorStepperPulse pulseA;
    MEIMotorStepperPulse pulseB;
    double               pulseWidth; /* output pulse width  (sec),
                                        10^(-7) < pulseWidth < 10^(-3) */
} MEIMotorStepper;
```

Change History: Modified in the 03.03.00

Description

MEIMotorStepper is a structure used to configure Stepper Motor parameters.

To use pulse outputs, you will need to:

1. Configure the motor type for STEPPER.
2. See the motor's stepper pulse width. Make sure it meets the drive's requirments and is not smaller than 2x the minimum pulse period.
3. Enable the motor's stepper loopback if there is no encoder feedback.
4. Configure the motor's stepper pulseA and pulseB types for STEP, DIR, CW, CCW, QUADA, or QUADB.
5. Select the motor I/O's config type for pulseA and pulseB. This will route the pulseA/B signals to the node's digital outputs.

loopback†	enables Step Loopback feature. When enabled, step output pulses counted to generate Actual Position. When disabled, external feedback device is required to generate actual position.
pulseA	Structure used to configure Step Pulse A.
pulseB	Structure used to configure Step Pulse B.
pulseWidth†	sets width of Step pulse. Valid values range from a minimum width of 0.1 usec to maximum width of 25.5 usec.

† - If you attempt to change this value and hardware is not connected, an error message will be returned.

See Also

MEIMotorStepperPulse | MEIMotorStepperPulseType

MEIMotorStepperPulse

Definition

```
typedef struct MEIMotorStepperPulse {
    MEIMotorStepperPulseType    type;
    MPI_BOOL                    invert;
} MEIMotorStepperPulse;
```

Change History: Modified in the 03.03.00

Description

MEIMotorStepperPulse is a structure used to configure Stepper Motor parameters.

To use pulse outputs, you will need to:

1. Configure the motor type for STEPPER.
2. See the motor's stepper pulse width. Make sure it meets the drive's requirments and is not smaller than 2x the minimum pulse period.
3. Enable the motor's stepper loopback if there is no encoder feedback.
4. Configure the motor's stepper pulseA and pulseB types for STEP, DIR, CW, CCW, QUADA, or QUADB.
5. Select the motor I/O's config type for pulseA and pulseB. This will route the pulseA/B signals to the node's digital outputs.

type	Configures the stepper motor's pulse type. If you attempt to change this value and hardware is not connected, an error message will be returned.
invert	If set to TRUE the actual Pulse output will be inverted by the FPGA. If you attempt to change this value and hardware is not connected, an error message will be returned.

See Also

MEIMotorStepper

MEIMotorStepperPulseType

Definition

```
typedef enum MEIMotorStepperPulseType {
    MEIMotorStepperPulseTypeSTEP,
    MEIMotorStepperPulseTypeDIR,

    MEIMotorStepperPulseTypeCW,
    MEIMotorStepperPulseTypeCCW,

    MEIMotorStepperPulseTypeQUADA,
    MEIMotorStepperPulseTypeQUADB,

} MEIMotorStepperPulseType;
```

Description

MEIMotorStepperPulseType is an enumeration used to specify the pulse output signal type.

MEIMotorStepperPulseTypeSTEP	This will enable the pulse output (either A or B) to generate a step output. Use it together with MEIMotorStepperPulseTypeDIR to provide a complete step interface.
MEIMotorStepperPulseTypeDIR	This will enable the pulse output (either A or B) to output the direction of the move. Use it together with MEIMotorStepperPulseTypeSTEP to provide a complete step interface.
MEIMotorStepperPulseTypeCW	This will enable the pulse output (either A or B) to output a clockwise pulse train. Used together with MEIMotorStepperPulseTypeCCW to provide a complete step interface.
MEIMotorStepperPulseTypeCCW	This will enable the pulse output (either A or B) to output a counter-clockwise pulse train. Use it together with MEIMotorStepperPulseTypeCW to provide a complete step interface.
MEIMotorStepperPulseTypeQUADA	This will enable the pulse output (either A or B) to output a quadrature signal (90 degree phase difference to MEIMotorStepperPulseTypeQUADB). Use it together with MEIMotorStepperPulseTypeQUADB to provide a complete quadrature interface.
MEIMotorStepperPulseTypeQUADB	This will enable the pulse output (either A or B) to output a quadrature signal (90 degree phase difference to MEIMotorStepperPulseTypeQUADA). Use it together with MEIMotorStepperPulseTypeQUADA to provide a complete quadrature interface.

See Also

MEIMotorStepperStatus

Definition

```
typedef struct MEIMotorStepperStatus {
   MPI_BOOL pulseLockLost; /* TRUE if Pulse jitter logic has lost lock,
                           otherwise FALSE */
   MPI_BOOL pulseStatus;   /* TRUE if a Pulse was generated
incorrectly,
                           otherwise FALSE */
} MEIMotorStepperStatus;
```

Change History: Modified in the 03.03.00

Description

MEIMotorStepperStatus is a structure used to check for error conditions relating to the pulse module.

pulseLockLost	Indicates that the pulse module lost timing lock with the SynqNet network.
pulseStatus	Indicates that the pulse module generated a new pulse when the previous pulse was not finished. It usually indicates that the pulse width is incorrect for the desired velocity.

See Also

MEIMotorStatus | MEIMotorStepper

MPIMotorType

Definition

```
typedef enum {
    MPIMotorTypeINVALID,

    MPIMotorTypeSERVO,
    MPIMotorTypeSTEPPER,
} MPIMotorType;
```

Description

MPIMotorType is an enumeration of valid Motor Types.

MPIMotorTypeSERVO	Motor configured as Servo
MPIMotorTypeSTEPPER	Motor configured as Stepper

See Also

mpiMotorConfigGet | mpiMotorConfigSet

mpiMotorEncoderFaultMaskBIT

Declaration

```
#define  mpiMotorEncoderFaultMaskBIT(fault)  (0x1 << (fault))
```

Required Header: stdmpi.h

Description

mpiMotorEncoderFaultMaskBIT converts the motor encoder fault into the motor encoder fault mask.

See Also

MPIMotorEncoderFault | MPIMotorEncoderFaultMask

MEIMotorAmpFaultsMAX

Definition

```
#define  MEIMotorAmpFaultsMAX  (32) /* max faults per motor */
```

Description

MEIMotorAmpFaultsMAX defines the maximum number of amp fault messages per motor.

See Also

MEIMotorAmpMsgMAX

Definition

```
#define   MEIMotorAmpMsgMAX    (200) /* max chars per motor */
```

Description

MEIMotorAmpMsgMAX defines the maximum number of characters per amp fault message.

See Also

MEIMotorAmpWarningsMAX

Definition

```
#define  MEIMotorAmpWarningsMAX  (32) /* max Warnings per motor */
```

Description

MEIMotorAmpWarningsMAX defines the maximum number of amp warning messages per motor.

See Also