Xmp Module

Introduction

The **Xmp** module provides the low-level interface between the controller and the MPI library. It defines the shared memory access between the controller's processor and the host CPU. It also contains several hardware constants and maximum values for resources. ALL data transactions between the MPI and controller are defined by this module.

The MPI provides a layer between the application code and the controller. It protects the application from *xmp.h* changes, hides controller complexity, handles semaphore locking, performs data validation and range checking, plus many other features. Normally, an application does not need the Xmp module. An application should ALWAYS use the MPI to access the controller. In some cases, the MPI may not have methods/structures to support a controller feature. For example, a custom feature may require support for direct access to the MPI, but it has not been yet been developed. In these cases, an application can use xmp.h and mpiControlMemoryGet/Set to directly access the controller.

WARNING!
The *xmp.h* file is version dependent. Make sure to ONLY use the xmp.h that was included with the MPI and controller firmware software package. Using mismatched xmp.h defines can cause unexpected and potentially dangerous behavior!

Be aware that the xmp.h is always changing. It is an internal file, used by the controller firmware and MPI. It is optimized for memory allocation and performance for the controller's processor. As new features are developed and improved, the xmp.h is modified. If you use xmp.h defines in your application code, make sure to check for changes when upgrading to new software releases.

Data Types

MEIXmp**CommMode**
MEIXmp**SwitchType**

MEIXmpCommMode

Declaration

```
typedef enum {
    MEIXmpCommModeNONE,
    MEIXmpCommModeCLOSED_LOOP,
    MEIXmpCommModeOPEN_LOOP,
    MEIXmpCommModeSIMULATE,
} MEIXmpCommMode;
```

Required Header: xmp.h

Description

MEIXmpCommMode is an enumeration that contains the different options for controller based sinusoidal commutation. It is important to understand that drive-based sinusoidal commutation is not covered in this enumeration. The value used for drive-based sinusoidal commutation is MEIXmpCommModeNONE.

Ways to tell if you are using drive or controller based sinusoidal commutation:

Drive-based Sinusoidal Commutation - only use MEIXmpCommModeNONE

- You are using a SynqNet drive.
- You use drive based commutation settings.
- You are using drive based commutation initialization.

Controller-based Sinusoidal Commutation

- You are not using a SynqNet drive.
- You are using a torque or current mode drive.
- You have two +/- 10 V current command inputs to your drive.
- You are using a SynqNet node that can provide two +/- 10 V outputs per motor (MEI-RMB-10V2, for example).
- You are using sinusoidal commutation.
- You drive does not provide sinusoidal commutation.
- You are not using brushed motors.
- Your drive is configured to accept two +/- 10V inputs for sinusoidal commutation.

Sinusoidal commutation requires initialization of some sort. If you are using Controller based sinusoidal commutation, please see Sinusoidal Commutation. There are three types of sinusoidal commutation initialization routines supported by the XMP/ZMP controller:

- Step (scstep.c)
- Hall (schall.c)
- Dither (scdither.c)

MEIXmpCommModeNONE	Default mode. This mode is used for: • All drives that perform their own commutation. • Any control loop that only requires a single command signal.
MEIXmpCommModeCLOSED_LOOP	Used during controller calculated sinusoidal commutation. This mode uses two command signals to command a two, three, or four phase sinusoidal commutation mode. This mode is most often used with an MEI RMB connected to a drive that has two ±10V inputs for torque command.
MEIXmpCommModeOPEN_LOOP	Used during sinusoidal commutation initialization before going to CLOSED_LOOP. This mode is only used when hall sensors are not available to go directly to CLOSED_LOOP mode.
MEIXmpCommModeSIMULATE	MEI internal mode to simulate a motor. Not used for actual motor control.

See Also

MEIXmpSwitchType

Declaration

```
typedef enum {
    MEIXmpSwitchTypeNONE,
    MEIXmpSwitchTypeMOTION_ONLY,
    MEIXmpSwitchTypeWINDOW,
    MEIXmpSwitchTypeUSER,
} MEIXmpSwitchType;
```

Required Header: xmp.h

Description

MEIXmpSwitchType is an enumeration for gain scheduling that determines the gain scheduling mode. Only MEIXmpSwitchTypeNONE and MEIXmpSwitchTypeMOTION_ONLY are available in standard firmware types.

Gain Scheduling is a feature that switches filter gains for the acceleration, deceleration, constant velocity, and idle states of motion. The post filters are not affected by gain scheduling. Standard algorithms are used with gain scheduling (PID, PIV). To change the gain scheduling type from *none* (uses only the gains in gain table index 0), use MEIFilterConfig. GainSwitchType is set with mpiFilterConfigSet(...).

When setting filter gain parameters using mpiFilterGainGet(...) and mpiFilterGainSet(...), use the gain index value to write to a gain index of your choosing.

MEIXmpSwitchTypeNONE	Default value in factory default firmware. This mode uses mpiFilterGainIndexSet() and mpiFilterGainIndexGet() to manipulate the gain index manually, if desired.
MEIXmpSwitchTypeMOTION_ONLY	Switch gains based on controller's switching algorithm.

MEIFilterGainIndex (go to MEIFilterGainIndex)

MEIFilterGainIndexNO_MOTION	When command velocity = 0
MEIFilterGainIndexACCEL	When command acceleration > 0
MEIFilterGainIndexDECEL	When command acceleration < 0
MEIFilterGainIndexVELOCITY	When command velocity > 0 and command acceleration = 0 The firmware automatically takes care of this switching. Be aware when checking the gain index, that the firmware can change the gain index in real time.

Description

Gain Scheduling is a feature that switches filter gains for the acceleration, deceleration, constant velocity, and idle states of motion. The post filters are not affected by gain scheduling. Standard algorithms are used with gain scheduling (PID, PIV). To change the gain scheduling type from NONE (uses only the gains in gain table index 0), use MEIFilterConfig.GainSwitchType, which is set with mpiFilterConfigSet(...).

Use mpiFilterConfigSet(...) to change MEIFilterConfig.GainSwitchType to one of the MEIXmpSwitchType enumerations to change the gain scheduling mode.

See Also

MPIFilterConfig | mpiFilterConfigGet | mpiFilterConfigSet | MEIFilterGainIndex | mpiFilterGainIndexSet | mpiFilterGainIndexGet | mpiFilterGainGet | mpiFilterGainSet