Global Objects

Introduction

Data types that are used by more than one module are defined in the mpidef.h header file.The definitions listed in this section are available to all modules, and are defined in mpidef.h.

Data Types

MPI**Action** / MEI**Action**
MEI**DataType**
MPI**IoSource**
MPI**IoType**
MPI**IoTrigger**
MEI**MaxBiQuadSections**
MPI**ModuleId** / MEI**ModuleId**
MEI**NetworkPort**
MEI**NetworkType**
MPI**State**
MPI**Status**
MEI**StatusFlag**
MPI**StatusMask** / MEI**StatusMask**
MPI**Trajectory**
MPI**Wait**

Macros

mpi**StatusMaskBIT**

MPIAction / MEIAction

MPIAction

```
typedef enum {
    MPIActionINVALID,

    MPIActionNONE,
    MPIActionSTOP,
    MPIActionE_STOP,
    MPIActionE_STOP_ABORT,
    MPIActionE_STOP_CMD_EQ_ACT,
    MPIActionABORT,

    MPIActionDONE,
    MPIActionSTART,
    MPIActionRESUME,
    MPIActionRESET,
    MPIActionCANCEL_REPEAT,

} MPIAction;
```

Description

MPIActionNONE	Performs no action. Use with MPIMotorEventConfig to prevent a motor event from performing an action.
MPIActionSTOP	Makes a motion supervisor perform a stop. This action can be commanded with mpiMotionAction(…) or by a motor event on the controller. Please see MPIMotionDecelTime for more information about stop actions.
MPIActionE_STOP	Makes a motion supervisor perform an e-stop. This action can be commanded with mpiMotionAction(…) or by a motor event on the controller. Please see MPIMotionDecelTime for more information about e-stop actions.
MPIActionE_STOP_ABORT	Makes a motion supervisor perform an e-stop and then an abort. This action can be commanded with mpiMotionAction(…) or by a motor event on the controller. Please see MPIMotionDecelTime for more information about e-stop actions.
MPIActionE_STOP_CMD_EQ_ACT	The command position is set equal to the previous sample's actual position. After the E_STOP time expires, the AmpEnable is disabled. The settling parameters are supported for this action, when settleOnEstopCmdEqAct in the MPIAxisConfig structure is enabled.
MPIActionABORT	Makes a motion supervisor perform an abort. This action can be commanded with mpiMotionAction(…) or by a motor event on the controller.

MPIActionDONE	is currently not supported and is reserved for future use.
MPIActionSTART	Intended to force a motion supervisor to start when it is waiting for some event (a delay or hold) before starting. This action is currently not supported.
MPIActionRESUME	Makes a motion supervisor to resume motion after a stop action has occurred. A motion supervisor can only resume a motion after a stop event, not an e-stop event. This action can be commanded with mpiMotionAction(…).
MPIActionRESET	Makes a motion supervisor return to an idle state after an error has occurred or after a stop, e-stop, abort, or e-stop/abort action has occurred. While abort actions and certain errors cause all associated motors to turn off their amp-enable lines, this action does not change the state of any amp-enable lines. One will have to call the method mpiMotorAmpEnableSet(…) to re-enable the amplifiers. This action can be commanded with mpiMotionAction(…).
MPIActionCANCEL_REPEAT	This action makes repeating cam finish at the end of the next cycle. i.e. The cam will continue executing until it passes the start of finish of the cam table. See Repeating Cams .

Remarks

MPIAction enumerations are used to perform some sort of action on an MPI object. Currently, only MPIMotion and MPIMotor use the MPIAction enumerations. One can command an MPIMotion object to perform some action with the mpiMotionAction(…) method, while one can get and set the types of actions that will be performed when certain motor events occur with the MPIMotorEventConfig structure with the mpiMotorEventConfigGet(…) and mpiMotorEventConfigSet(…) methods.

An MPIAction can be generated from the host or the firmware. Below is a table where MPIActions originate (start):

MPIAction	Originating from Host	Originating from XMP Firmware
Start	mpiMotionAction(…) (currently unsupported)	NEVER
Resume	mpiMotionAction(…)	NEVER
Reset	mpiMotionAction(…)	NEVER
Stop	mpiMotionAction(…)	Event
E_Stop	mpiMotionAction(…)	Event
ABORT	mpiMotionAction(…)	Event
DONE	NEVER	NEVER

MEIAction

```
typedef enum {
    MEIActionMAP = MPIActionLAST,

} MEIAction;
```

Description

MEIActionMAP	MotionAction will write the axis mapping relationship of the motion supervisor to the controller. This mapping is written automatically when mpiMotionStart() is called.

See Also

mpiMotionAction | MPIMotionDecelTime | MPIMotorEventConfig
mpiMotorEventConfigGet | mpiMotorEventConfigSet | MPIEvent | MPIAxisConfig

MEIDataType

```
typedef enum {
    MEIDataTypeINVALID = 0, /* this should stay zero - static arrays init
                               to zero by default */
    MEIDataTypeCHAR,
    MEIDataTypeSHORT,
    MEIDataTypeUSHORT,
    MEIDataTypeLONG,
    MEIDataTypeULONG,
    MEIDataTypeFLOAT,
    MEIDataTypeDOUBLE,
} MEIDataType;

static MEIDataType  MEIFilterGainTypePID[MPIFilterCoeffCOUNT_MAX] =
{
    MEIDataTypeFLOAT, /* Kp               */
    MEIDataTypeFLOAT, /* Ki               */
    MEIDataTypeFLOAT, /* Kd               */

    MEIDataTypeFLOAT, /* Kpff             */
    MEIDataTypeFLOAT, /* Kvff             */
    MEIDataTypeFLOAT, /* Kaff             */
    MEIDataTypeFLOAT, /* Kfff             */

    MEIDataTypeFLOAT, /* MovingIMax       */
    MEIDataTypeFLOAT, /* RestIMax         */

    MEIDataTypeLONG,  /* DRate            */

    MEIDataTypeFLOAT, /* OutputLimit      */
    MEIDataTypeFLOAT, /* OutputLimitHigh  */
    MEIDataTypeFLOAT, /* OutputLimitLow   */
    MEIDataTypeFLOAT, /* OutputOffset     */
    MEIDataTypeFLOAT, /* Ka0              */
    MEIDataTypeFLOAT, /* Ka1              */
    MEIDataTypeFLOAT, /* Ka2              */
};
```

Description

DataType is an enumeration of data types for the filter coefficients.

MEIDataTypeCHAR	character filter data type
MEIDataTypeSHORT	short integer filter data type
MEIDataTypeUSHORT	unsigned short integer filter data type
MEIDataTypeLONG	long integer filter data type
MEIDataTypeULONG	unsigned long filter data type
MEIDataTypeFLOAT	floating point filter data type
MEIDataTypeDOUBLE	double precision floating point filter data type

See Also

MPIIoSource

MPIIoSource

```
typedef union {
    MPIHandle   motor;  /* MOTOR */
    long        index;  /* USER */
} MPIIoSource;
```

Description

motor	Handle to a motor object that is the source for the IO.
index	Value of the index for a user input. User IO's are no longer supported by the xmp (user IO's are handled through the motor object).

See Also

MPIIoType

MPIIoType

```
typedef enum {
    MPIIoTypeINVALID,

    MPIIoTypeMOTOR,
    MPIIoTypeUSER,

} MPIIoType;
```

Description

MPIIoTypeMotor	Value specifies the IO type as motor (the IO source is a motor object).
MPIIoTypeUSER	Value specifies the IO type as user (User IO types are currently not supported. User IO is available through the motor objects).

See Also

MPIIoTrigger

MPIIoTrigger

```
typedef struct MPIIoTrigger {
    MPIIoType        type;
    MPIIoSource      source;
    unsigned long    mask;
    unsigned long    pattern;
} MPIIoTrigger;
```

Description

type	see MPIIoType.
source	see MPIIoSource.
mask	Value that specifies the mask to be applied to the IO.
pattern	Value that specifies the pattern to be compared to the masked IO.

See Also

MEIMaxBiQuadSections

MEIMaxBiQuadSections

```
#define MEIMaxBiQuadSections    (6)
```

Description

MaxBiQuadSections defines the maximum number of biquad sections in an MPIFilter's postfilter. Postfilters are used to digitally filter the output of a control loop. One common use for postfilters is the compensation of system resonances.

NOTE: The PIV algorithm uses the last biquad section internally and so the user can only use (MEIMaxBiQuadSections-1) sections if the PIV algorithm is the current control algorithm.

See Also

meiFilterPostfilterGet | meiFilterPostfilterSet | meiFilterPostfilterSectionGet | meiFilterPostfilterSectionSet

MPIModuleId / MEIModuleId

MPIModuleId

```
typedef enum {
    MPIModuleIdINVALID,


    MPIModuleIdMESSAGE,
    MPIModuleIdAXIS,
    MPIModuleIdCAPTURE,
    MPIModuleIdCOMMAND,
    MPIModuleIdCOMPARE,
    MPIModuleIdCOMPENSATOR,
    MPIModuleIdCONTROL,

    MPIModuleIdEVENT,
    MPIModuleIdEVENTMGR,
    MPIModuleIdFILTER,

    MPIModuleIdMOTION,
    MPIModuleIdMOTOR,

    MPIModuleIdNOTIFY,
    MPIModuleIdPATH,
    MPIModuleIdPROBE,
    MPIModuleIdRECORDER,
    MPIModuleIdSEQUENCE,


    MPIModuleIdEXTERNAL = 0x80,

    MPIModuleIdMAX = 0xFF
} MPIModuleId;
```

Description

ModuleId is used to identify what module a particular MPIHandle belongs to. If the handle is an external memory pointer instead of an MPI object handle, MPIModuleIdEXTERNAL will be returned by MPI methods.

MEIModuleId

```
typedef enum {
    MEIModuleIdPLATFORM,


    MEIModuleIdCAN,

    MEIModuleIdCLIENT,
    MEIModuleIdELEMENT,
    MEIModuleIdFLASH,
    MEIModuleIdLIST,
    MEIModuleIdMAP,
    MEIModuleIdPACKET,
    MEIModuleIdSERVER,

    MEIModuleIdSYNQNET,
    MEIModuleIdSQNODE,
    MEIModuleIdDRIVE_MAP,



    MEIModuleIdBLOCK = MEIModuleIdSQNODE,

}MEIModuleId;
```

Description

ModuleId is used to identify what module a particular MPIHandle belongs to. If the handle is an external memory pointer instead of an MPI object handle, MPIModuleIdEXTERNAL will be returned by MPI methods.

See Also

mpiObjectModuleId | mpiObjectValidate

MEINetworkPort

MEINetworkPort

```
typedef enum MEINetworkPort {
    MEINetworkPortIN0,
    MEINetworkPortOUT0,
} MEINetworkPort;
```

Description

MEINetworkPort enumerations are used to specify a network port. Network ports represent the physical network connections into which the cables are plugged. These ports are commonly found in pairs on SynqNet controllers and SynqNet nodes. OUT ports from the controller or an upstream node connect to the IN port of a downstream node. When the OUT port of the last downstream node is connected back to the IN port of the controller, the network is considered to be configured as a RING network type.

MEINetworkPortIN0	"IN" Network port 0
MEINetworkPortOUT0	"OUT" Network port 0

See Also

MEINetworkType

MEINetworkType

MEINetworkType

```
typedef enum MEINetworkType {
    MEINetworkTypeINVALID = -1,   /* no nodes found */
    MEINetworkTypeSTRING,
    MEINetworkTypeSTRING_TERMINATED,
    MEINetworkTypeRING,
} MEINetworkType;
```

Description

The **NetworkType** enumeration lists all network topologies supported by this MPI release.

NOTE: Due to the un-terminated nature of the MEINetworkTypeSTRING network type discovering a topology of this type requires a timeout period to detect the end of the string. Thus discovering a string topology will take a longer time than other networks types.

MEINetworkTypeINVALID	No nodes were found on the network
MEINetworkTypeSTRING	The network topology type is a string of nodes
MEINetworkTypeSTRING_TERMINATED	The network topology type is a string of nodes. The last node on the string is terminated.
MEINetworkTypeRING	The network topology type is a ring of nodes.

See Also

MEISynqNetInfo | meiSynqNetInfo

MPIState

MPIState

```
typedef enum {
    MPIStateIDLE,
    MPIStateMOVING,
    MPIStateSTOPPING,
    MPIStateSTOPPED,
    MPIStateSTOPPING_ERROR,
    MPIStateERROR,
} MPIState;
```

Description

State is an MPI enum that is used to describe the current state of the controller's motion state machine. MPIState resides in the MPIStatus structure. Currently MPIState is only used with motion module.

MPIStateIDLE	The state of motion is idle and waiting to resume motion.
MPIStateMOVING	The state of the motion is moving.
MPIStateSTOPPING	The state of the motion is stopping. This occurs from a Stop event, but not an E_Stop, E_Stop Abort, or Abort events. The stop command could have come from the firmware or MPI.
MPIStateSTOPPED	The move has stopped due to a STOP command. It requires a MotionReset to go back to IDLE OR a MotionResume to resume stopped motion. MotionStart can also be called at any time to start a new move.
MPIStateSTOPPING_ERROR	The state of the motion is performing an emergency stop and/or abort on all axes. The move is stopping due to an error (ESTOP, ABORT, etc…).
MPIStateERROR	The state of the motion is in error. The error state is generated from an E_Stop or Abort event. It requires a MotionReset to get back to IDLE before loading another move.

See Also

MPIStatus

MPIStatus

MPIStatus

```
typedef struct MPIStatus {
    MPIState        state;
    MPIAction       action;
    MPIEventMask    eventMask;

    long    settled;
    long    atTarget;

    MPIStatusMask    statusMask;
} MPIStatus;
```

Description

state	Value that indicates the state of an xmp controller's motion supervisor.
action	Value that indicates the action to perform for a motion supervisor.
eventMask	Array that defines the event mask bits. The array is defined as typedef MPIEventMaskELEMENT_TYPE MPIEventMask[MPIEventMaskELEMENTS].
settled	Value that indicates if an axis associated with a motion supervisor has settled (is in fine position).
atTarget	Value that indicates if an axis associated with a motion supervisor has completed its command trajectory (i.e. the command position has reached the targeted end point of the move).
statusMask	Value is an enumeration of bit masks for the MEIStatusFlags. The status masks represent the present condition for an object.

See Also

MPIState | MPIStatusMask

MEIStatusFlag

MEIStatusFlag

```
typedef enum {
    MEIStatusFlagBROKEN_WIRE,                /* 1 */
    MEIStatusFlagILLEGAL_STATE,              /* 2 */
    MEIStatusFlagABS_ENCODER_FAULT,          /* 3 */
    MEIStatusFlagABS_ENCODER_TIMEOUT,        /* 4 */
    MEIStatusFlagBROKEN_WIRE_SECONDARY,      /* 5 */
    MEIStatusFlagILLEGAL_STATE_SECONDARY,    /* 6 */
} MEIStatusFlag;
```

Description

StatusFlag is an enumeration of status flags (bits) for an object. The status flags represent the present condition for an object.

Please see MEIStatusMask data type for more information.

See Also

MEIStatusMask | MPIStatusMask

MPIStatusMask / MEIStatusMask

MPIStatusMask

```
typedef enum {
    MPIStatusMaskNONE   = 0x0,

    MPIStatusMaskMOTOR  = MPIStatusMaskNONE, /* 0x00000001 */

    MPIStatusMaskALL    = mpiStatusMaskBIT(MPIStatusFlagLAST) - 1
                                          /* 0x00000001 */
} MPIStatusMask;
```

Description **StatusMask** is an enumeration of bit masks for the MEIStatusFlags. The status masks represent the present condition for an object.

MPIStatusMaskMOTOR	Value specifies the motor's status mask.
MPIStatusMaskALL	Value specifies the status mask that encompasses all the possible status flags.

See Also MPIStatus

MEIStatusMask

```
typedef enum {
    MEIStatusMaskBROKEN_WIRE             =
mpiStatusMaskBIT(MEIStatusFlagBROKEN_WIRE),            /* 0x00000002 */
    MEIStatusMaskILLEGAL_STATE           =
mpiStatusMaskBIT(MEIStatusFlagILLEGAL_STATE),          /* 0x00000004 */
    MEIStatusMaskABS_ENCODER_FAULT       =
mpiStatusMaskBIT(MEIStatusFlagABS_ENCODER_FAULT),      /* 0x00000008 */
    MEIStatusMaskABS_ENCODER_TIMEOUT     =
mpiStatusMaskBIT(MEIStatusFlagABS_ENCODER_TIMEOUT),    /* 0x00000010 */
    MEIStatusMaskBROKEN_WIRE_SECONDARY   =
mpiStatusMaskBIT(MEIStatusFlagBROKEN_WIRE_SECONDARY),  /* 0x00000020 */
    MEIStatusMaskILLEGAL_STATE_SECONDARY =
mpiStatusMaskBIT(MEIStatusFlagILLEGAL_STATE_SECONDARY), /* 0x00000040 */

    MEIStatusMaskMOTOR =   /* 0x0000001E */
                          (MEIStatusMaskBROKEN_WIRE  |
                           MEIStatusMaskILLEGAL_STATE |
                           MEIStatusMaskABS_ENCODER_FAULT |
                           MEIStatusMaskABS_ENCODER_TIMEOUT),

    MEIStatusMaskALL   =   /* 0x0000001E */
```

```
        (mpiStatusMaskBIT(MEIStatusFlagLAST) - 1) & ~MPIStatusMaskALL
} MEIStatusMask;
```

Description

StatusMask is an enumeration of bit masks for the MEIStatusFlags. The status masks represent the present condition for an object.

MEIStatusMaskBROKEN_WIRE	Broken wire on the primary encoder input signals. Occurs when any of the differential encoder input channels (A+ and A-, B+ and B-, or I+ and I-), have the same logic state. This mask indicates either a floating or shorted encoder input signal.
MEIStatusMaskILLEGAL_STATE	Illegal encoder logic state on the primary encoder input signals. Occurs when the A and B encoder input channels transition simultaneously. This mask indicates either faulty encoder signal logic, encoder frequencies that are too high, or noisy encoder signals.
MEIStatusMaskABS_ENCODER_FAULT	Absolute encoder initialization failure. Occurs when the hardware fails to read the absolute position information from an encoder.
MEIStatusMaskABS_ENCODER_TIMEOUT	Absolute encoder response timeout. Occurs when the encoder fails to respond to a request for absolute position data.
MEIStatusMaskBROKEN_WIRE_SECONDARY	Broken wire on the secondary encoder input signals. Occurs when any of the differential encoder input channels (A+ and A-, B+ and B-, or I+ and I-), have the same logic state. This mask indicates either a floating or shorted encoder input signal.
MEIStatusMaskILLEGAL_STATE_SECONDARY	Illegal encoder logic state on the secondary encoder inputs. Occurs when the A and B encoder input channels transition simultaneously. This flag indicates either faulty encoder signal logic, encoder frequencies that are too high, or noisy encoder signals.
MEIStatusMaskMOTOR	Bit mask containing all of the motor specific MEIStatusFlags set.
MEIStatusMaskALL	Bit mask containing all of the MEIStatusFlags set.

See Also

MPIStatus | MEIStatusFlag | MPIStatusFlag | MPIStatusMask

MPITrajectory

MPITrajectory

```
typedef struct MPITrajectory {
    double    velocity;
    double    acceleration;
    double    deceleration;
    double    jerkPercent;
    double    accelerationJerk;
    double    decelerationJerk;
} MPITrajectory;
```

Description

The **Trajectory** structure contains the motion profile parameters for simple point to point type motion.

NOTE: Not all firmware binaries support S_CURVE_JERK and VELOCITY_JERK motion types due to code space limitations. mpiMotionStart(…) and mpiMotionModify(…) will return MPIMotionMessagePROFILE_NOT_SUPPORTED if the controller does not support the requested move type.

velocity	Rate of change of position. Specifies the constant slew rate for S_CURVE, TRAPEZOIDAL, S_CURVE_JERK, VELOCITY, and VELOCITY_JERK motion types. Units are counts per second.
acceleration	Rate of change of velocity. Specifies the initial ramp to reach constant velocity for TRAPEZOIDAL and VELOCITY motion types. Also, specifies the ramp from the initial jerk to the next jerk before constant velocity for S_CURVE, S_CURVE_JERK and VELOCITY_JERK motion types. Units are counts per second * second.
deceleration	Rate of change of velocity. Specifies the final ramp to reach zero velocity for TRAPEZOIDAL and VELOCITY motion types. Also, specifies the ramp from the jerk after constant velocity to the final jerk for S_CURVE and S_CURVE_JERK motion types. Units are counts per second * second.
jerkPercent	Portion of acceleration and deceleration ramp to perform jerk profile for S_CURVE, VELOCITY, S_CURVE_JERK, and VELOCITY_JERK motion types. Units are in percent. Range is 0.0 to 100.0.
accelerationJerk	Rate of change of acceleration. Specifies the initial jerk rates to reach constant velocity for S_CURVE_JERK and VELOCITY_JERK motion types. Units are counts per second * second * second.
decelerationJerk	Rate of change of deceleration. Specifies the final jerk rate to reach zero velocity for S_CURVE_JERK motion types. Units are counts per second * second * second.

Sample Code

```
MPITrajectory trajectory;

    mpiAxisTrajectory(axis, &trajectory);

    printf("Velocity %.3f\n"
           "Acceleration %.3f\n",
           trajectory.velocity,
           trajectory.acceleration);
```

See Also

MPIMotionType | mpiMotionTrajectory | mpiMotionStart | mpiMotionModify

MPIWait

MPIWait

```
typedef enum {
    MPIWaitFOREVER = -1,
    MPIWaitPOLL = 0,
    MPIWaitMSEC
} MPIWait;
```

Description

Wait enumerations define basic wait times for certain MPI methods.

MPIWaitFOREVER	Makes MPI methods wait forever for an event to occur before returning.
MPIWaitPOLL	Makes MPI methods see if a certain event has occurred. If an event has not occurred, then the MPI method will generally return immediately returning the value MPIMessageTIMEOUT.
MPIWaitMSEC	Defines a period of one millisecond. If used alone, this will make MPI methods wait for one millisecond for an event occurs before returning. One can pass an an agument a multiple of MPIWaitMSEC to make MPI methods wait longer periods of time. For example, the following statement will make mpiPlatformKey wait 5 milliseconds for a user keystroke: mpiPlatformKey(5 * MPIWaitMSEC); If an event does not occur within the specified time, MPI methods will generally return the value MPIMessageTIMEOUT.

WARNING

The MPI depends on the ability of the operating system it is running on to be able to activate threads or put threads to sleep for a specified period of time in order for these times to be accurate. Microsoft Windows platforms are not real-time operating systems and are known to be unable to activate threads any quicker than 10 milliseconds. If you encounter a timing problem, it is likely an operating system timing issue.

See Also

mpiControlInterruptWait | mpiNotifyEventWait | mpiObjectTimeoutGet
mpiObjectTimeoutSet | meiPlatformKey

mpiStatusMaskBIT

Declaration

```
#define   mpiStatusMaskBIT(flag)        (0x1 << (flag))
```

Required Header stdmpi.h

Description **StatusMaskBIT** converts the status flag into the status mask.

See Also MPIStatusMask