

# Probe Objects

## Introduction

A Probe object manages a hardware logic block. The Probe hardware can latch and store up to 16 motor positions or node clock values within one controller sample period. The latches can be triggered by one of the configurable input sources (home, index, +/- limits, or one of the first 16 general purpose motor I/O).

The Probe is used in applications that need multiple data captures within one controller sample period. For applications that do not need multiple latches per sample, the Capture object is a much better solution. For example, a homing routine should use Capture (not Probe). Homing usually requires one or two precise position latches triggered by a home sensor and/or encoder index input to calibrate the position feedback to a physical location. The Probe is useful in high speed scanning applications, where the input can trigger at rates higher than the controller's sample rate. In this case, the Probe can be used with the Data Recorder to collect bursts of scan data. Later, the application can process the data.

Probe data is in raw position feedback counts or SynqNet node clock ticks. Typically, the high-speed Probe data and the sampled position data are collected with the Data Recorder. Later, the data is processed and the probed positions are calculated or interpolated from the time values.

There are two Probe data types: Position and Time. For the position data type, the lower 16 bits of the position counters are latched in the FPGA. This methodology works well for incremental quadrature encoders. For the time data type, the FPGA latches the clock value. The application must read the position from the controller's previous sample and the present sample, to interpolate the probe position. This methodology works well for cyclic feedback data that is digitally transmitted from the drive to the FPGA. Many drives have proprietary serial encoders that decode the encoder position and send the position information to the FPGA once per sample. In these cases, time-based probing is more accurate than position based probing.

For the position data type, the trigger source input and the feedback must be on the same motor.

For the time data type, the trigger source input and the clock must be on the same node. But, since SynqNet nodes are synchronized to the controller's clock, the position values can be interpolated across multiple nodes. Thus, a single probe input used with the Data Recorder can be used to collect/interpolate positions for several axes, across several nodes.

| [Error Messages](#) |

## Methods

[mpiProbeConfigGet](#)

[mpiProbeConfigSet](#)

[mpiProbeControl](#)

[mpiProbeCreate](#)

[mpiProbeDelete](#)

[meiProbeInfo](#)

[mpiProbeParams](#)

[mpiProbeStatus](#)

[mpiProbeValidate](#)

## Data Types

[MEIProbeAddress](#)

[MPIProbeConfig](#)

[MPIProbeData](#)

[MPIProbeInfo](#)

[MPIProbeMessage](#) / [MEIProbeMessage](#)

[MPIProbeNumber](#)

[MPIProbeParams](#)

[MPIProbeSource](#)

[MPIProbeStatus](#)

[MEIProbeTrace](#)

[MPIProbeType](#)

## Constants

[MPIProbeMaxProbeRegisters](#)

# mpiProbeConfigGet

## Declaration

```
long mpiProbeConfigGet(MPIProbe      probe,
                       MPIProbeConfig *config,
                       void          *external);
```

**Required Header:** stdmpi.h

## Description

**mpiProbeConfigGet** reads a Probe's configuration and writes it into the structure pointed to by **config**, and also into the implementation specific structure pointed to by **external** (if external is not NULL).

## Remarks

**external** is reserved for future functionality and should be set to NULL.

<b>probe</b>	a handle to a Probe object.
<b>*config</b>	a pointer to a Probe configuration structure.
<b>*external</b>	a pointer to an implementation specific structure.

## Return Values

[MPIMessageOK](#)

[MPIProbeMessagePROBE\\_INVALID](#)

## See Also

[mpiProbeConfigSet](#)

# mpiProbeConfigSet

## Declaration

```
long mpiProbeConfigSet(MPIProbe      probe ,
                       MPIProbeConfig *config ,
                       void          *external ) ;
```

**Required Header:** stdmpi.h

## Description

**mpiProbeConfigSet** sets a Probe's 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).

## Remarks

**external** is reserved for future functionality and should be set to NULL.

<b>probe</b>	a handle to a Probe object.
<b>*config</b>	a pointer to a Probe configuration structure.
<b>*external</b>	a pointer to an implementation specific structure. <b>external</b> is reserved for future functionality and should be set to NULL.

## Return Values

[MPIMessageOK](#)

[MPIProbeMessagePROBE\\_TYPE\\_INVALID](#)

[MPIProbeMessagePROBE\\_INVALID](#)

## See Also

[mpiProbeConfigGet](#)

# mpiProbeControl

## Declaration

```
MPIControl mpiProbeControl(MPIProbe probe);
```

Required Header: stdmpi.h

## Description

**mpiProbeControl** returns a handle to the Control object with which the Probe object is associated.

<b>probe</b>	a handle to a Probe object.
--------------	-----------------------------

### Return Values

<b>MPIControl</b>	handle to a Control object.
-------------------	-----------------------------

<b>MPIHandleVOID</b>	if Probe is not valid
----------------------	-----------------------

## See Also

[mpiProbeCreate](#) | [mpiControlCreate](#)

# mpiProbeCreate

## Declaration

```
MPIProbe mpiProbeCreate(MPIProbe control ,
                        MPIProbeParams *params )
```

**Required Header:** stdmpi.h

## Description

**mpiProbeCreate** creates a Probe object identified by the params, which is associated with a control object. ProbeCreate is the equivalent of a C++ constructor.

The probe params structure specifies the probe type, a probe object identification number, and an index to a hardware probe engine. See [MPIProbeParams](#) for details.

<b>control</b>	a handle to a Probe object.
<b>*params</b>	a pointer to a probe parameter's structure.

Return Values	
<b>handle</b>	handle to a Probe object.
<b>MPIHandleVOID</b>	if the object could not be created.
<b>MPIProbeMessagePROBE_INVALID</b>	The probe number specified was invalid. The likely cause of this is that the node you are referring to does not support the probe feature. Please refer to the <a href="#">Node FPGA Images: Features Table</a> to see if your node supports probe.

## See Also

[mpiProbeDelete](#) | [mpiProbeValidate](#) | [MPIProbeParams](#)

# mpiProbeDelete

## Declaration

```
long mpiProbeDelete(MPIProbe probe);
```

**Required Header:** stdmpi.h

## Description

**mpiProbeDelete** deletes a Probe object and invalidates its handle. ProbeDelete is the equivalent of a C++ constructor.

## Remarks

All objects that are created must be deleted in the reverse order to avoid memory leaks.

<b>probe</b>	a handle to a Probe object.
--------------	-----------------------------

## Return Values

[MPIMessageOK](#)

## See Also

[mpiProbeCreate](#) | [mpiProbeValidate](#)

# meiProbeInfo

## Declaration

```
long meiProbeInfo(MPIProbe probe,  
                 MEIProbeInfo *info);
```

Required Header: stdmei.h

## Description

**meiProbeInfo** reads a Probe's information and writes it into a structure pointed to by *info*. The *info* structure contains read only data about the probe engine.

<b>probe</b>	a handle to a Probe object.
<b>*info</b>	contains read only data about the probe engine.

## Return Values

[MPIMessageOK](#)

## See Also



# mpiProbeParams

## Declaration

```
long mpiProbeParams( MPIProbe      probe ,  
                    MPIProbeParams *params ) ;
```

**Required Header:** stdmpi.h

## Description

**mpiProbeParams** reads the parameters of a Probe object and writes it to the structure pointed to by *params*. The Probe parameters are set with [mpiProbeCreate\(...\)](#).

<b>probe</b>	a handle to a Probe object.
<b>*params</b>	a pointer to a Probe parameter's structure.

## Return Values

[MPIMessageOK](#)

## See Also

[mpiProbeStatus](#) | [mpiProbeCreate](#)

# mpiProbeStatus

## Declaration

```
long mpiProbeStatus(MPIProbe      probe ,
                   MPIProbeStatus *status ,
                   void          *external ) ;
```

**Required Header:** stdmpi.h

## Description

**mpiProbeStatus** reads status from the Probe and writes it into the structure pointed to by **status** and also writes it into the implementation-specific structure pointed to be **external** (if external is not NULL). The Probe status structure contains data from the probe engine, including digital I/O states and the latched probe values.

**external** is reserved for future functionality and should be set to NULL.

<b>probe</b>	a handle to a Probe object.
<b>*status</b>	a pointer to a Probe status structure.
<b>*external</b>	a pointer to an implementation specific structure. <b>external</b> is reserved for future functionality and should be set to NULL.

## Return Values

[MPIMessageOK](#)

## See Also

[meiProbeInfo](#) | [mpiProbeParams](#)

# mpiProbeValidate

## Declaration

```
long mpiProbeValidate(MPIProbe probe) ;
```

**Required Header:** stdmpi.h

## Description

**mpiProbeValidate** validates the Probe object and its handle. ProbeValidate should be called immediately after an object is created.

<b>probe</b>	a handle to a Probe object.
--------------	-----------------------------

## Return Values

[MPIMessageOK](#)

## See Also

[mpiProbeCreate](#) | [mpiProbeDelete](#)

# MEIProbeAddress

## Definition

```
typedef struct MEIProbeAddress {  
    long    *primaryPosition;  
    long    *secondaryPosition;  
    long    *time;  
    long    *status;  
    short   *data;  
} MEIProbeAddress;
```

## Description

**MEIProbeAddress** contains the controller addresses for the Probe data fields. The addresses in this structure are useful for configuring the Recorder to collect data during probing.

<b>*primaryAddress</b>	a pointer to the address for the Motor's primary position feedback.
<b>*secondaryPosition</b>	a pointer to the address for the Motor's secondary position feedback.
<b>*time</b>	a pointer to the address for the feedback time value.
<b>*status</b>	a pointer to the address for the Probe status
<b>*data</b>	a pointer to the address for the Probe data.

## See Also

[MEIProbeInfo](#)

# MPIProbeConfig

## Definition

```
typedef struct MPIProbeConfig {
    MPI_BOOL      enable;          /* TRUE/FALSE */
    MPIProbeSource source;
    MPIProbeData  data;
    MPI_BOOL      inputFilter;    /* TRUE/FALSE */
} MPIProbeConfig;
```

**Change History:** Modified in the 03.03.00

## Description

**MPIProbeConfig** specifies the Probe's configuration.

<b>enable</b>	Enables or disables the Probe triggering. A value of TRUE enables the Probe triggering, FALSE disables Probe triggering.
<b>source</b>	An enumerated Probe trigger source input. A Probe can be configured to trigger from one input source.
<b>data</b>	An enumerated Probe data type. A probe can be configured to collect position or time data from a motor's feedback.
<b>inputFilter</b>	<p>Enables or disables the source input filtering. The hardware Probe engine has a 4 state digital filter to eliminate noise on the input signal. When enabled, the Probe will not trigger until the input signal is stable for 4 clock periods.</p> <p>For example, the MEI-RMB clock is 25Mhz. When the inputFilter is enabled, the input signal must transition to and remain at either a high or low state for 160 nanoseconds to trigger the Probe.</p>

## See Also

[meiProbeConfigGet](#) | [meiProbeConfigSet](#)

# MPIProbeData

## Definition

```
typedef enum MPIProbeData {  
    MPIProbeDataPOSITION_PRIMARY,  
    MPIProbeDataPOSITION_SECONDARY,  
    MPIProbeDataTIME,  
} MPIProbeData;
```

## Description

**MPIProbeData** is an enumeration of Probe data types. This specifies to the probe engine what data to collect when triggered.

<b>MPIProbeDataPOSITION_PRIMARY</b>	Position from the motor's primary feedback
<b>MPIProbeDataPOSITION_SECONDARY</b>	Position from the motor's secondary feedback
<b>MPIProbeDataTIME</b>	Clock value from the node. The clock value can be used to derive the position by interpolating between the positions from the previous sample and the present sample period and dividing by the difference between the probe clock value and the initial clock value.

## See Also

[meiProbeConfigGet](#) | [meiProbeConfigSet](#) | [MPIProbeConfig](#)

# MEIProbeInfo

## Definition

```
typedef struct MEIProbeInfo {  
    MEIProbeAddress    address;  
} MEIProbeInfo;
```

## Description

**MEIProbeInfo** contains read only information about the Probe.

<b>address</b>	a structure containing controller addresses for useful data during probing.
----------------	---

## See Also

[meiProbeInfo](#) | [meiProbeStatus](#)

# MPIProbeMessage / MEIProbeMessage

## Definition: MPIProbeMessage

```
typedef enum {
    MPIProbeMessageNODE_INVALID,
    MPIProbeMessagePROBE_TYPE_INVALID,
    MPIProbeMessagePROBE_INVALID,
} MPIProbeMessage;
```

**Change History:** Modified in the 03.03.00

## Description

**MPIProbeMessage** is an enumeration of SynqNet node error messages that can be returned by the MPI library.

### MPIProbeMessageNODE\_INVALID

The SynqNet node number is not available on the network. This message code is returned by MPI methods that fail a service command transaction due to the specified node number being greater than or equal to the total number of nodes discovered during network initialization. To correct this problem, check the discovered node count with [meiSynqNetInfo\(...\)](#). If the node count is not what you expected, check your network wiring, node condition, and re-initialize the network with [mpiControlReset\(...\)](#).

### MPIProbeMessagePROBE\_TYPE\_INVALID

The Probe data type is not valid. This message is returned by [mpiProbeConfigSet\(...\)](#) if the specified Probe data type is not one of the enumerated values.

### MPIProbeMessagePROBE\_INVALID

The Probe is not valid. This message is returned by [mpiProbeConfigGet\(...\)](#) or [mpiProbeConfigSet\(...\)](#) if the specified Probe params or configurations are out of range. To correct this problem, check your Probe params and config structures.

## Definition: MEIProbeMessage

```
typedef enum {
    MEIProbeMessageLAST = MPIProbeMessageLAST
} MEIProbeMessage;
```



**Change History:** Modified in the 03.03.00

## Description

**MEIProbeMessage** is mainly a placeholder.

## See Also

# MPIProbeNumber

## Definition

```
typedef union MPIProbeNumber {  
    long      motor;  
    long      node;  
} MPIProbeNumber;
```

## Description

**MPIProbeNumber** specifies the identification number for the Probe. The Probe type determines the identification number definition.

<b>motor</b>	An index to identify the motor that is associated with the Probe. Must be specified for MPIProbeTypeMOTOR Probe types.
<b>node</b>	An index to identify the node that is associated with the Probe. Must be specified for MPIProbeTypeIO Probe types.

## See Also

[MPIProbeType](#) | [MPIProbeParams](#)

# MPIProbeParams

## Definition

```
typedef struct MPIProbeParams {  
    MPIProbeType      type ;  
    MPIProbeNumber  number ;  
    long            probeIndex ;  
} MPIProbeParams ;
```

## Description

**MPIProbeParams** specifies the parameters for a Probe engine.

<b>type</b>	An enumerated Probe type. Determines the Probe number definition.
<b>number</b>	An union to identify the object number that is associated with the Probe. The Probe type defines the number.
<b>probeIndex</b>	An index to specify the Probe engine. The hardware may support one or more Probe engines per Motor or Node. The number of Probe engines supported by the hardware can be read with <a href="#">meiMotorInfo(...)</a> and <a href="#">meiSqNodeInfo(...)</a> .

## See Also

[mpiProbeParams](#) | [mpiProbeCreate](#) | [MEIMotorInfo](#) | [MEISqNodeInfo](#)

# MPIProbeSource

## Definition

```
typedef enum MPIProbeSource {
    MPIProbeSourceHOME,
    MPIProbeSourceINDEX,
    MPIProbeSourceLIMIT_HW_NEG,
    MPIProbeSourceLIMIT_HW_POS,
    MPIProbeSourceINDEX_SECONDARY,
    MPIProbeSourceMOTOR_IO_0,
    MPIProbeSourceMOTOR_IO_1,
    MPIProbeSourceMOTOR_IO_2,
    MPIProbeSourceMOTOR_IO_3,
    MPIProbeSourceMOTOR_IO_4,
    MPIProbeSourceMOTOR_IO_5,
    MPIProbeSourceMOTOR_IO_6,
    MPIProbeSourceMOTOR_IO_7,
    MPIProbeSourceMOTOR_IO_8,
    MPIProbeSourceMOTOR_IO_9,
    MPIProbeSourceMOTOR_IO_10,
    MPIProbeSourceMOTOR_IO_11,
    MPIProbeSourceMOTOR_IO_12,
    MPIProbeSourceMOTOR_IO_13,
    MPIProbeSourceMOTOR_IO_14,
    MPIProbeSourceMOTOR_IO_15,
} MPIProbeSource;
```

## Description

**MPIProbeSource** is an enumeration of input trigger sources for a Probe.

<b>MPIProbeSourceHOME</b>	Home input in the dedicated I/O
<b>MPIProbeSourceINDEX</b>	Index input from the primary encoder in the dedicated I/O
<b>MPIProbeSourceLIMIT_HW_NEG</b>	Hardware Negative Limit input in the dedicated I/O
<b>MPIProbeSourceLIMIT_HW_POS</b>	Hardware Positive Limit input in the dedicated I/O
<b>MPIProbeSourceINDEX_SECONDARY</b>	Index input from the secondary encoder in the dedicated I/O

<b>MPIProbeSourceMOTOR_IO_0</b>	Bit number 0 in the Motor's configurable I/O
<b>MPIProbeSourceMOTOR_IO_1</b>	Bit number 0 in the Motor's configurable I/O
<b>MPIProbeSourceMOTOR_IO_2</b>	Bit number 0 in the Motor's configurable I/O
<b>MPIProbeSourceMOTOR_IO_3</b>	Bit number 0 in the Motor's configurable I/O
<b>MPIProbeSourceMOTOR_IO_4</b>	Bit number 0 in the Motor's configurable I/O
<b>MPIProbeSourceMOTOR_IO_5</b>	Bit number 0 in the Motor's configurable I/O
<b>MPIProbeSourceMOTOR_IO_6</b>	Bit number 0 in the Motor's configurable I/O
<b>MPIProbeSourceMOTOR_IO_7</b>	Bit number 0 in the Motor's configurable I/O
<b>MPIProbeSourceMOTOR_IO_8</b>	Bit number 0 in the Motor's configurable I/O
<b>MPIProbeSourceMOTOR_IO_9</b>	Bit number 0 in the Motor's configurable I/O
<b>MPIProbeSourceMOTOR_IO_10</b>	Bit number 0 in the Motor's configurable I/O
<b>MPIProbeSourceMOTOR_IO_11</b>	Bit number 0 in the Motor's configurable I/O
<b>MPIProbeSourceMOTOR_IO_12</b>	Bit number 0 in the Motor's configurable I/O
<b>MPIProbeSourceMOTOR_IO_13</b>	Bit number 0 in the Motor's configurable I/O
<b>MPIProbeSourceMOTOR_IO_14</b>	Bit number 0 in the Motor's configurable I/O
<b>MPIProbeSourceMOTOR_IO_15</b>	Bit number 0 in the Motor's configurable I/O

## See Also

[MPIProbeConfig](#)

# MPIProbeStatus

## Definition

```
typedef struct MPIProbeStatus {
    long    maxRegisters;
    long    index;
    long    io;
    long    regs[MPIMaxProbeRegisters];
} MPIProbeStatus;
```

## Description

**MPIProbeStatus** specifies the present condition of the Probe engine.

<b>maxRegisters</b>	The maximum number of Probe data registers that are available. This value is determined by the hardware Probe engine capability and the number of data registers that are initialized in the SynqNet packets.
<b>index</b>	The Probe engine index. Depending on the Probe type, the Probe is either associated with a motor or node object. Each Probe can have 1 or more hardware Probe engines. The probe engine is identified by its index.
<b>io</b>	The input state values for the Probe data. Each bit represents a triggered Probe data value in the regs array. If a bit is active, then a corresponding Probe data value was stored in the regs array.  For example, if io = 0x3, then regs[0] and regs[1] have valid Probe data.
<b>regs</b>	An array of captured Probe data. The maximum number of data fields is specified by maxRegisters. Each element of the array that contains valid Probe data is flagged by a bit in the I/Ostatus.

## See Also

[mpiProbeStatus](#) | [mpiProbeParams](#) | [MPIProbeParams](#)

# MEIProbeTrace

## Definition

```
typedef enum {  
    MEIProbeTraceFIRST = MEITraceLAST << 1,  
    MEIProbeTraceLAST  = MEIProbeTraceFIRST << 15  
} MEIProbeTrace;
```

## Description

**MEIProbeTrace** is an enumeration of probe object trace bits to enable debug tracing.

## See Also

# MPIProbeType

## Definition

```
typedef enum MPIProbeType {  
    MPIProbeTypeMOTOR,  
    MPIProbeTypeIO,  
} MPIProbeType;
```

## Description

**MPIProbeType** is an enumeration of Probe types. This specifies whether the Probe engine is associated with a Motor or Node I/O. The available Probe type is dependent on the hardware Probe engine implementation.

<b>MPIProbeTypeMOTOR</b>	The Probe hardware engine is a component of the Motor object. For the Motor type, the Probe is identified by the Motor number.
<b>MPIProbeTypeIO</b>	The Probe hardware engine is a component of the Node I/O. For the I/O type, the Probe is identified by the Node number.

## See Also

[MPIProbeNumber](#) | [MPIProbeParams](#)



# MPIMaxProbeRegisters

## Definition

```
#define MPIMaxProbeRegisters    (16)
```

## Description

**MPIMaxProbeRegisters** defines the maximum limit for the probe registers for the Probe hardware.

## See Also