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.

Methods

mpiProbe**ConfigGet**
mpiProbe**ConfigSet**
mpiProbe**Control**
mpiProbe**Create**
mpiProbe**Delete**
meiProbe**Info**
mpiProbe**Params**
mpiProbe**Status**
mpiProbe**Validate**

Data Types

MEIProbe**Address**
MPIProbe**Config**
MPIProbe**Data**
MPIProbe**Info**
MPIProbe**Message** / MEIProbe**Message**
MPIProbe**Number**
MPIProbe**Params**
MPIProbe**Source**
MPIProbe**Status**
MEIProbe**Trace**
MPIProbe**Type**

Constants

MPIProbe**MaxProbeRegisters**

mpiProbeConfigGet

Declaration

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

Required Header

stdmei.h

Description

ProbeConfigGet 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).

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

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

Return Values

MPIMessageOK	if *ProbeConfigGet* successfully reads the Probe configuration from the controller.
MPIProbeMessagePROBE_INVALID	if the specified Probe params or configurations are out of range.

See Also

mpiProbeConfigSet

mpiProbeConfigSet

Declaration

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

Required Header

stdmei.h

Description

ProbeConfigSet 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).

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

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

Return Values

MPIMessageOK	if *ProbeConfigSet* successfully wrote the Probe configuration to the controller.
MPIProbeMessagePROBE_TYPE_INVALID	if the specified Probe data type is not one of the enumerated values
MPIProbeMessagePROBE_INVALID	if the specified Probe params or configurations are out of range.

See Also

mpiProbeConfigGet

mpiProbeControl

Declaration

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

Required Header

stdmei.h

Description

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

probe	a handle to a Probe object.

Return Values

MPIControl	handle to a Control object.
MPIHandleVOID	if Probe is not valid

See Also

mpiProbeCreate | mpiControlCreate

mpiProbeDelete

Declaration

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

Required Header

stdmei.h

Description

ProbeDelete deletes a Probe object and invalidates its handle.

ProbeDelete is the equivalent of a C++ constructor.

probe	a handle to a Probe object.

Remarks

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

Return Values

MPIMessageOK	if *ProbeDelete* successfully deletes a Probe object and invalidates its handle.

See Also

mpiProbeCreate | mpiProbeValidate

meiProbeInfo

Declaration

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

Required Header

stdmei.h

Description

ProbeControl 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.

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

Return Values

MPIMessageOK	

See Also

mpiProbeParams

Declaration

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

Required Header

stdmei.h

Description

ProbeParams reads the parameters of a Probe object and writes it to the structure pointed to by params. The Probe parameters are set with mpiProbeCreate(…).

probe	a handle to a Probe object.
***params**	a pointer to a Probe parameter's structure.

Return Values

MPIMessageOK	

See Also

mpiProbeStatus | mpiProbeCreate

mpiProbeStatus

Declaration

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

Required Header

stdmei.h

Description

ProbeStatus 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.

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

Return Values

MPIMessageOK	

See Also

meiProbeInfo | mpiProbeParams

mpiProbeValidate

Declaration

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

Required Header

stdmei.h

Description

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

probe	a handle to a Probe object.

Return Values

MPIMessageOK	if Probe is a handle to a valid object.

See Also

mpiProbeCreate | mpiProbeDelete

MEIProbeAddress

MEIProbeAddress

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

Description

The **ProbeAddress** structure 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.

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

See Also

MEIProbeInfo

MPIProbeConfig

MPIProbeConfig

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

Description

The **ProbeConfig** structure specifies the Probe's configuration.

enable	Enables or disables the Probe triggering. A value of TRUE enables the Probe triggering, FALSE disables Probe triggering.
source	An enumerated Probe trigger source input. A Probe can be configured to trigger from one input source.
data	An enumerated Probe data type. A probe can be configured to collect position or time data from a motor's feedback.
inputFilter	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. 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.

See Also

meiProbeConfigGet | meiProbeConfigSet

MPIProbeData

MPIProbeData

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

Description

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

MPIProbeDataPOSITION_PRIMARY	Position from the motor's primary feedback
MPIProbeDataPOSITION_SECONDARY	Position from the motor's secondary feedback
MPIProbeDataTIME	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

mpiProbeCreate

Declaration

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

Required Header

stdmei.h

Description

ProbeCreate 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.

control	a handle to a Probe object.
***params**	a pointer to a probe parameter's structure.

Return Values

handle	handle to a Probe object.
MPIHandleVOID	if the object could not be created.

See Also

mpiProbeDelete | mpiProbeValidate | MPIProbeParams

MEIProbeInfo

MEIProbeInfo

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

Description

The **ProbeInfo** structure contains read only information about the Probe.

address	a structure containing controller addresses for useful data during probing.

See Also

meiProbeInfo | meiProbeStatus

MPIProbeMessage / MEIProbeMessage

MPIProbeMessage

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

Description

MPISequenceMessageNODE_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(…).

MPISequenceMessagePROBE_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.

MPISequenceMessagePROBE_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.

See Also

MEIProbeMessage

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

Description

ProbeMessage is mainly a placeholder.

See Also

MPIProbeNumber

MPIProbeNumber

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

Description

The ProbeNumber union specifies the identification number for the Probe. The Probe type determines the identification number definition.

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

See Also

MPIProbeType | MPIProbeParams

MPIProbeParams

MPIProbeParams

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

Description

The **ProbeParams** structure specifies the parameters for a Probe engine.

type	An enumerated Probe type. Determines the Probe number definition.
number	An union to identify the object number that is associated with the Probe. The Probe type defines the number.
probeIndex	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 meiMotorInfo(…) and meiSqNodeInfo(…).

See Also

mpiProbeParams | mpiProbeCreate | MEIMotorInfo | MEISqNodeInfo

MPIProbeSource

MPIProbeSource

```
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

ProbeSource is an enumeration of input trigger sources for a Probe.

MPIProbeSourceHOME	Home input in the dedicated I/O
MPIProbeSourceINDEX	Index input from the primary encoder in the dedicated I/O
MPIProbeSourceLIMIT_HW_NEG	Hardware Negative Limit input in the dedicated I/O
MPIProbeSourceLIMIT_HW_POS	Hardware Positive Limit input in the dedicated I/O
MPIProbeSourceINDEX_SECONDARY	Index input from the secondary encoder in the dedicated I/O
MPIProbeSourceMOTOR_IO_0	Bit number 0 in the Motor's configurable I/O

MPIProbeStatus

MPIProbeStatus

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

Description

The **ProbeStatus** structure specifies the present condition of the Probe engine.

maxRegisters	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.
index	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.
io	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.
regs	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

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

See Also

MPIProbeConfig | MEIMotorIoMask

MEIProbeTrace

MEIProbeTrace

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

Description

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

See Also

MPIProbeType

MPIProbeType

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

Description

ProbeType 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.

MPIProbeTypeMOTOR	The Probe hardware engine is a component of the Motor object. For the Motor type, the Probe is identified by the Motor number.
MPIProbeTypeIO	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

MPIMaxProbeRegisters

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

Description

The Probe hardware has a maximum limit for the probe registers, which is defined by **MaxProbeRegisters**.

See Also