

Capture Objects

Introduction

A **Capture** object manages a single position capture logic block. It represents the physical hardware capture logic and data. When configured and armed, the capture logic block can latch a motor's position based on one or more source input triggers.

The Capture object's number, motor input trigger sources, edge, type, feedback source, and capture index are all configurable. There are two capture types: Position and Time based. For the Position type, the position counters are latched in the FPGA and are read directly by the controller. This methodology works well for incremental quadrature encoders. For the Time type, the FPGA latches the clock and the controller reads the clock value and position value for that sample period. The controller interpolates the position value from the previous sample's position, the present sample's position, and the clock data. This methodology works very well for cyclic feedback data that is digitally transmitted from the drive to the FPGA. Many drives have a proprietary serial encoder that decodes the encoder position and sends the position information to the FPGA once per sample. In these cases, time-based capture is more accurate than position-based capture.

For the **Position** type, the motor number for the input sources and the feedback motor number must be the same.

For the **Time** type, the motor number and feedback motor number can be different. This makes it possible to use inputs from one node to capture positions on another node.

When using captures, the controller must have enough enabled captures to process the specified capture number. The controller will process the enabled captures (captureCount) every sample period. Since each capture object is configurable, use the minimum number of captures possible for best controller performance. For example, if you want to use 2 captures for motor 0 and motor 3, set the capture count to 2 and use capture number 0 and 1.

NOTE: Time-based capture will only work correctly if the speed of an axis is less than 344 million counts per second.

For an overview of the Capture Engine, see [diagram](#) below.

See Also: [Overview of Capture](#)

| [Error Messages](#) |

Methods

Create, Delete, Validate Methods

<u>mpiCaptureCreate</u>	Create Capture object
<u>mpiCaptureDelete</u>	Delete Capture object
<u>mpiCaptureValidate</u>	Validate Capture object

Configuration and Information Methods

<u>mpiCaptureConfigGet</u>	Get Capture configuration
<u>mpiCaptureConfigSet</u>	Set Capture configuration
<u>mpiCaptureStatus</u>	Get status of Capture
<u>mpiCaptureConfigReset</u>	

Action Methods

<u>mpiCaptureArm</u>	Arm capture object
--------------------------------------	--------------------

Memory Methods

<u>mpiCaptureMemory</u>	Set address to Capture memory
<u>mpiCaptureMemoryGet</u>	Copy Capture memory to application memory
<u>mpiCaptureMemorySet</u>	Copy application memory to Capture memory

Relational Methods

<u>mpiCaptureControl</u>	
<u>mpiCaptureNumber</u>	Get index of Capture (for Control list)

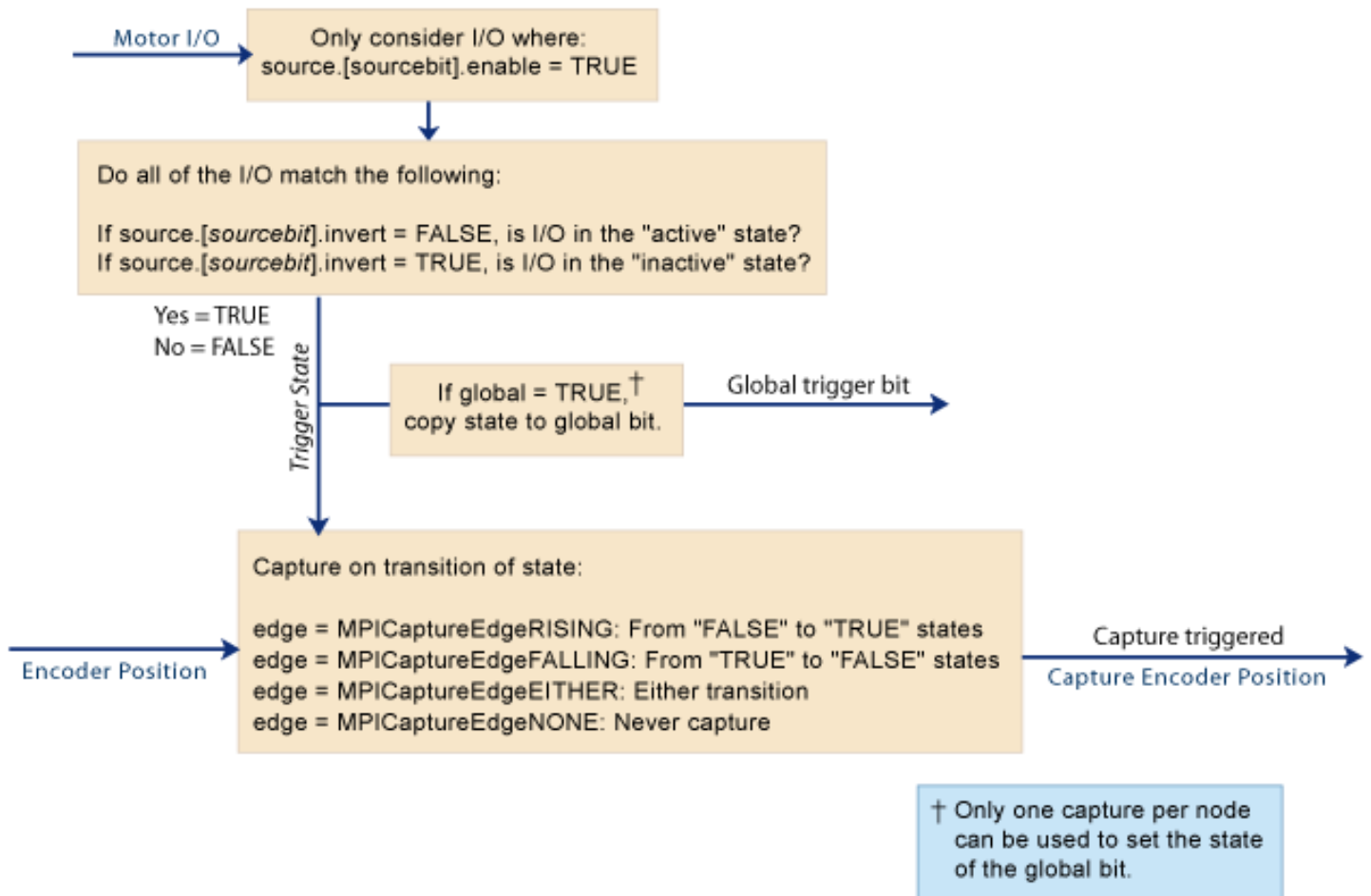
Data Types

[MPICaptureConfig](#)
[MPICaptureEdge](#)
[MPICaptureMessage](#) / [MEICaptureMessage](#)
[MPICaptureSource](#)
[MPICaptureState](#)
[MPICaptureStatus](#)
[MPICaptureTrigger](#)
[MPICaptureTriggerGlobal](#)
[MPICaptureType](#)

Constants

[MPCaptureNOT_MAPPED](#)

Capture Engine Diagram



mpiCaptureCreate

Declaration

```
MPICapture mpiCaptureCreate(MPIControl control,
                             long number);
```

Required Header: stdmpi.h

Description

mpiCaptureCreate creates a Capture object. The Capture object is identified by its association with a motor object, the motor's encoder and the encoder's capture number. The maximum number of enabled captures is 32.

CaptureCreate is the equivalent of a C++ constructor.

control	a handle to a Control object
number	An index to the encoder's capture block.

Return Values

handle	to a Capture object
MPIHandleVOID	if the object could not be created

See Also

[mpiCaptureNumber](#)

mpiCaptureNumber

Declaration

```
long mpiCaptureNumber(MPICapture capture,  
                     long *number)
```

Required Header: stdmpi.h

Description

mpiCaptureNumber reads the index of the capture block associated with the capture object and writes it into the contents of a long pointed to by encoder.

capture	a handle to a capture object
*number	pointer to the capture number.

Return Values

[MPIMessageOK](#)

See Also

[mpiCaptureCreate](#)

mpiCaptureDelete

Declaration

```
long mpiCaptureDelete(MPICapture capture)
```

Required Header: stdmpi.h

Description

mpiCaptureDelete deletes a Capture object and invalidates its handle (*capture*).

CaptureDelete is the equivalent of a C++ destructor.

Return Values	
MPIMessageOK	

See Also

[mpiCaptureCreate](#) | [mpiCaptureValidate](#)

mpiCaptureValidate

Declaration

```
long mpiCaptureValidate(MPICapture capture)
```

Required Header: stdmpi.h

Description

mpiCaptureValidate validates the Capture object and its handle. CaptureValidate should be called immediately after an object is created.

capture	a handle to a Capture object
----------------	------------------------------

Return Values

[MPIMessageOK](#)

See Also

[mpiCaptureCreate](#) | [mpiCaptureDelete](#)

mpiCaptureConfigGet

Declaration

```
long mpiCaptureConfigGet(MPICapture capture,
                        MPICaptureConfig *config,
                        void *external)
```

Required Header: stdmpi.h

Description

mpiCaptureConfigGet gets a Capture object's (*capture*) 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 a Capture object's configuration information in *external* is in addition to the Capture object's configuration information in *config*, i.e, the Capture object's configuration information in *config* and in *external* is not the same information. Note that *config* or *external* can be NULL (but not both NULL).

If a capture has been previously configured (non-default), use `mpiCaptureConfigReset(...)` to return the capture to the default configuration before calling `mpiCaptureConfigGet(...)` and `mpiCaptureConfigSet(...)`. Or if you do not call `mpiCaptureConfigReset(...)`, make sure that all members of the `MPICaptureConfig{...}` structure are explicitly set before calling `mpiCaptureConfigSet(...)`.

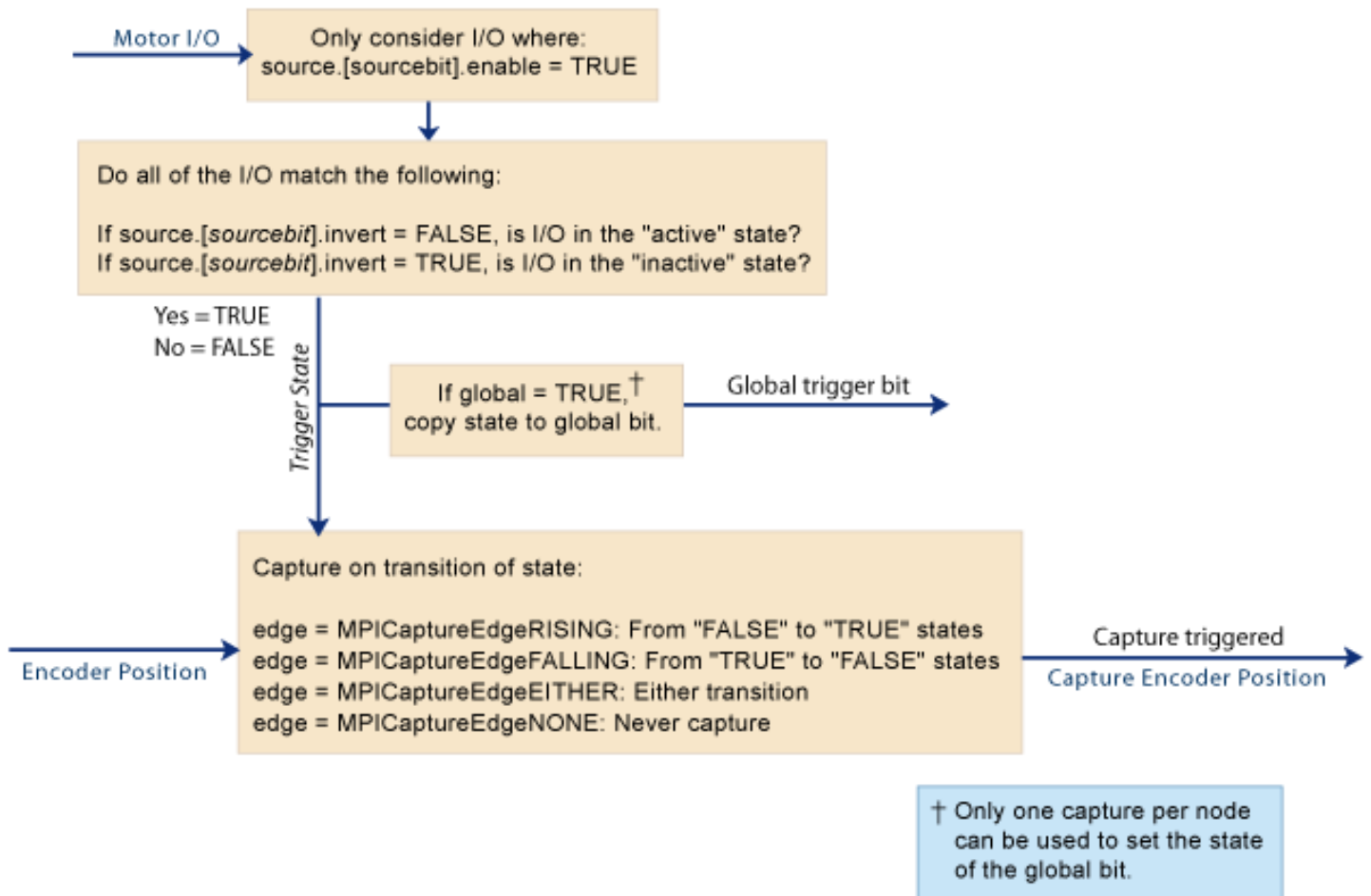
Remarks

**external* should be NULL.

Return Values

[MPIMessageOK](#)

Capture Engine Diagram



See Also

[mpiCaptureConfigSet](#) | [mpiCaptureConfigReset](#)

mpiCaptureConfigSet

Declaration

```
long mpiCaptureConfigSet(MPICapture      capture,
                        MPICaptureConfig *config,
                        void                *external)
```

Required Header: stdmpi.h

Description

mpiCaptureConfigSet sets a Capture object's (*capture*) 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 Capture object's configuration information in *external* is *in addition* to the Capture object's configuration information in *config*, i.e, the Capture object's configuration information in *config* and in *external* is not the same information. Note that *config* or *external* can be NULL (but not both NULL).

If a capture has been previously configured (non-default), use `mpiCaptureConfigReset(...)` to return the capture to the default configuration before calling `mpiCaptureConfigGet(...)` and `mpiCaptureConfigSet(...)`. Or if you do not call `mpiCaptureConfigReset(...)`, make sure that all members of the `MPICaptureConfig{...}` structure are explicitly set before calling `mpiCaptureConfigSet(...)`.

NOTE:

Don't reconfigure the source (trigger) capture resource with different settings via different `MPICapture` objects. Time-based capture allows you to capture multiple encoder positions using the same trigger. Currently, each motor only has one trigger resource— i.e. one trigger whose trigger state may be configured. If `MPICapture` object 0 is configured to trigger off of motor 0's index line, and then `MPICapture` object 1 is configured to trigger off of motor 0's home input, then only the capture trigger for motor 0 will have been reconfigured. Both `MPICapture` object 0 and `MPICapture` object 1 will now trigger off of motor 0's home input.

Remarks

**external* should be NULL.

Return Values	
MPIMessageOK	

See Also

[mpiCaptureConfigGet](#) | [mpiCaptureConfigReset](#)

mpiCaptureConfigReset

Declaration

```
long mpiCaptureConfigReset(MPICapture capture);
```

Required Header: stdmpi.h

Description

mpiCaptureConfigGet return the capture object to its unmapped state.

A capture object has no assumed resources, and is unmapped under default conditions. When a capture is first created, its captureMotorNumber and feedbackMotorNumber are unmapped. Once a capture has been configured, the next time that the capture object is created, it will retain the captureMotorNumber and feedbackMotorNumber that was previously assigned. mpiCaptureConfigReset(...) will return the capture object to its unmapped state.

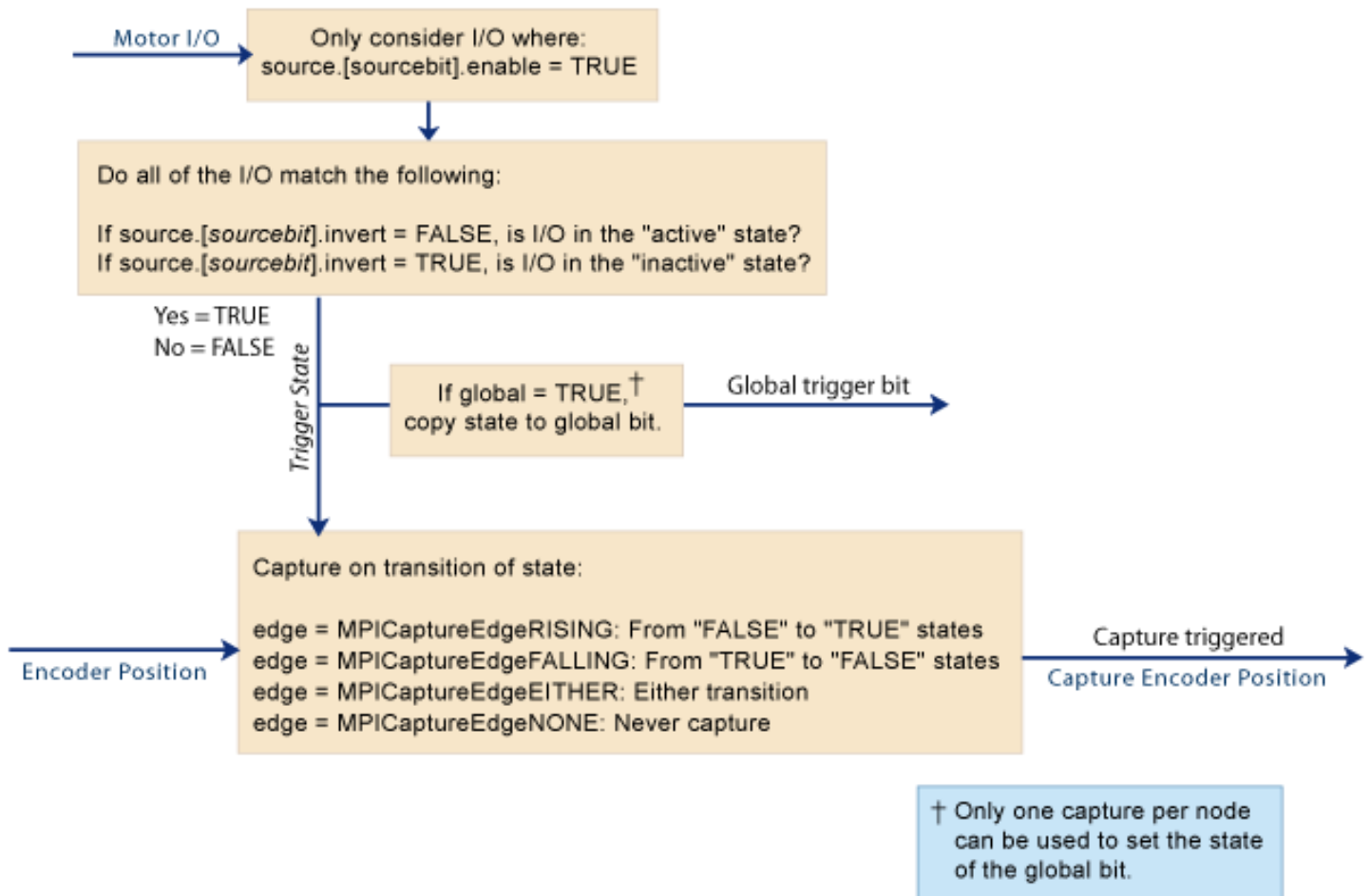
If a capture has been previously configured (non-default), use mpiCaptureConfigReset(...) to return the capture to the default configuration before calling mpiCaptureConfigGet(...) and mpiCaptureConfigSet(...). Or if you do not call mpiCaptureConfigReset(...), make sure that all members of the MPICaptureConfig{...} structure are explicitly set before calling mpiCaptureConfigSet(...).

capture	a handle to a Capture object
----------------	------------------------------

Return Values

[MPIMessageOK](#)

Capture Engine Diagram



See Also

[mpiCaptureConfigGet](#) | [mpiCaptureConfigSet](#) | [MPICaptureConfig](#)

mpiCaptureStatus

Declaration

```
long mpiCaptureStatus(MPICapture      capture ,
                    MPICaptureStatus *status ,
                    void              *external )
```

Required Header: stdmpi.h

Description

mpiCaptureStatus writes a Capture object's (**capture**) status into the structure pointed to by ***status**, 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 always be set to NULL.

capture	a handle to a Capture object
*status	a pointer to MPIStatus structure
*external	a pointer to an implementation-specific structure

Return Values

[MPIMessageOK](#)

[MPIMessageARG_INVALID](#)

See Also

mpiCaptureArm

Declaration

```
long mpiCaptureArm(MPICapture      capture ,
                  MPI_BOOL      arm )
```

Required Header: stdmpi.h

Change History: Modified in the 03.03.00

Description

mpiCaptureArm arms or disarms *capture*.

<i>Value of "arm"</i>	<i>Action of mpiCaptureArm</i>
FALSE	Disarms <i>capture</i> and sets the state of <i>capture</i> to MPICaptureStateIDLE
TRUE	Arms <i>capture</i> and sets the state of <i>capture</i> to MPICaptureStateARMED

Return Values

[MPIMessageOK](#)

See Also

[MPICaptureState](#)

mpiCaptureMemory

Declaration

```
long mpiCaptureMemory(MPICapture capture,
                     void **memory)
```

Required Header: stdmpi.h

Description

mpiCaptureMemory writes an address [which is used to access a Capture object's (*capture*) memory] to the contents of *memory*. This address, or an address calculated from it, can be passed as the *src* parameter to `mpiCaptureMemoryGet(...)` and as the *dst* parameter to `mpiCaptureMemorySet(...)`.

Return Values	
MPIMessageOK	

See Also

[mpiCaptureMemoryGet](#) | [mpiCaptureMemorySet](#)

mpiCaptureMemoryGet

Declaration

```
long mpiCaptureMemoryGet(MPICapture capture,
                        void *dst,
                        const void *src,
                        long count)
```

Required Header: stdmpi.h

Change History: Modified in the 03.03.00

Description

mpiCaptureMemoryGet copies *count* bytes of a Capture object's (*capture*) memory (starting at address *src*) and writes them into application memory (starting at address *dst*).

Return Values

[MPIMessageOK](#)

See Also

[mpiCaptureMemory](#) | [mpiCaptureMemorySet](#)

mpiCaptureMemorySet

Declaration

```
long mpiCaptureMemorySet(MPICapture capture,
                          void *dst,
                          const void *src,
                          long count)
```

Required Header: stdmpi.h

Change History: Modified in the 03.03.00

Description

mpiCaptureMemorySet copies count bytes of application memory (starting at address **src**) and writes them into a Capture object's (**capture**) memory (starting at address **dst**).

Return Values

[MPIMessageOK](#)

See Also

[mpiCaptureMemory](#) | [mpiCaptureMemoryGet](#)

mpiCaptureControl

Declaration

```
long mpiCaptureControl(MPICapture capture);
```

Required Header: stdmpi.h

Change History: Added in the 03.02.00

Description

mpiCaptureControl returns a handle to the motion controller (Control) with which the Capture is associated.

capture	a handle to a Capture object
----------------	------------------------------

Return Values

MPIControl	Handle to a Control object
-------------------	----------------------------

MPIHandleVOID	If capture is invalid
----------------------	-----------------------

See Also

[mpiCaptureCreate](#) | [mpiControlCreate](#)

MPICaptureConfig

Definition

```
typedef struct MPICaptureConfig {
    MPICaptureTrigger      source[MPICaptureSourceCOUNT];
                          /* use MPICaptureSource to index */
    MPICaptureEdge       edge;
    MPICaptureTriggerGlobal global;
    MPICaptureType       type;
    long                  captureMotorNumber;
    long                  feedbackMotorNumber; /* the same as
                          captureMotorNumber for POSITION capture */
    MPIMotorEncoder     encoder;
    long                  captureIndex;      /* 0,1,... */
} MPICaptureConfig;
```

Description

source [MPICaptureSourceCOUNT]	An array of capture trigger source inputs. The capture can be configured to trigger from one or more sources. See MPICaptureTrigger and MPICaptureSourceCOUNT .
edge	An enumerated index to the trigger edge type. The capture can be configured to trigger from a variety of logic. See MPICaptureEdge .
global	A structure to configure the global capture, to chain capture block triggering. See MPICaptureTriggerGlobal .
type	Specifies either position-based or time-based capture. Use MPICaptureTypePOSITION for position-based capture and MPICaptureTypeTIME for time-based capture.
captureMotorNumber	The number of the motor whose "source" (MPICaptureTrigger) is used to capture position.
feedbackMotorNumber	The number of the motor whose position is being returned from the capture event. (It must be the same as <code>captureMotorNumber</code> for position capture).
encoder	Specifies the encoder feedback being captured.

captureIndex

A zero-based index that specifies which capture resource on an axis is to be associated with the capture object.

Each axis on a node has a given number of captures associated with it. An axis may have up to 4 capture resources on it. At present, no vendor provides a node with more than one capture resource, therefore, **captureIndex must be set to zero.**

Remarks

Time-based capture will only work correctly if the speed of an axis is less than 344 million counts per second.

See Also

[MPIOCaptureType](#) | [mpiCaptureConfigGet](#) | [mpiCaptureConfigSet](#)

MPICaptureStatus

Definition

```
typedef struct MPICaptureStatus {
    MPICaptureState    state;
    double              latchedValue;
} MPICaptureStatus;
```

Description

state	An enumerated value representing the present state of the capture logic
latchedValue	<p>The captured encoder position value. This value is only valid when the state is CAPTURED.</p> <p>The actual position value is the captured encoder position value subtracted by the origin variable.</p> <p>The origin variable can be set/get through mpiAxisOriginSet(...) and mpiAxisOriginGet(...).</p> <p>Recall that encoder positions have no origin adjustment whereas actual positions do have origin adjustment.</p> <p>Please refer to Using the Origin Variable for more information.</p>

Sample Code

```
void displayPosition(MPICapture    capture,
                    MPIAxis       axis)
{
    double          origin;
    MPICaptureStatus captureStatus;

    /* Check captured position */
    mpiCaptureStatus(capture,
                    &captureStatus,
                    NULL);

    /* Getting origin variable and store it to origin */
    mpiAxisOriginGet(axis, &origin);

    if (captureStatus.state == MPICaptureStateCAPTURED)
```

```
{  
    printf("Latched actual position: %.0lf\n",  
          captureStatus.latchedValue - origin);  
    printf("Latched encoder position: %.0lf\n",  
          captureStatus.latchedValue);  
}  
}
```

See Also

[MPICaptureState](#)

MPICaptureState

Definition

```
typedef enum {  
    MPICaptureStateIDLE,  
    MPICaptureStateARMED,  
    MPICaptureStateCAPTURED,  
    MPICaptureStateCLEAR,  
} MPICaptureState;
```

Description

MPICaptureStateIDLE	Capture is not armed. This is the default state.
MPICaptureStateARMED	Capture is armed, but has not triggered yet.
MPICaptureStateCAPTURED	Capture triggered and position data is valid.
MPICaptureStateCLEAR	Capture is not armed, but has not transitioned to the IDLE state yet. This is an internal transitional state between CAPTURED and IDLE. It occurs when a capture is disarmed.

See Also

[MPICaptureStatus](#)

MPICaptureTrigger

Definition

```
typedef struct MPICaptureTrigger {  
    MPI_BOOL enabled;  
    MPI_BOOL invert;  
} MPICaptureTrigger;
```

Change History: Modified in the 03.03.00

Description

The **MPICaptureTrigger** structure specifies the trigger configurations for a capture.

enabled	Enables or disables the trigger. A value of TRUE enables the trigger, FALSE disables the trigger.
invert	Normal or inverted trigger polarity. A value of FALSE indicates normal polarity, TRUE indicates inverted polarity.

See Also

[MPICaptureSource](#)

MPICaptureSource

Definition

```
typedef enum MPICaptureSource {  
    MPICaptureSourceMOTOR_IO_0,  
    MPICaptureSourceMOTOR_IO_1,  
    MPICaptureSourceMOTOR_IO_2,  
    MPICaptureSourceMOTOR_IO_3,  
    MPICaptureSourceMOTOR_IO_4,  
    MPICaptureSourceMOTOR_IO_5,  
    MPICaptureSourceMOTOR_IO_6,  
    MPICaptureSourceMOTOR_IO_7,  
    MPICaptureSourceHOME,  
    MPICaptureSourceINDEX,  
    MPICaptureSourceLIMIT_HW_NEG,  
    MPICaptureSourceLIMIT_HW_POS,  
    MPICaptureSourceGLOBAL,  
    MPICaptureSourceINDEX_SECONDARY,  
    MPICaptureSourceCOUNT,  
} MPICaptureSource;
```

Description

MPICaptureSource is an enumeration of input trigger sources for a capture.

When using one of the `MPICaptureSourceMOTOR_IO` values in `MPICaptureSource`, you can determine which `MPICaptureSourceMOTOR_IO` to use by referencing the appropriate node module. Look in `NodeMotorIoConfig` (replacing *Node* with your node name) in the appropriate node module. Add the appropriate `NodeMotorIoConfig` value to `MPICaptureSourceMOTOR_IO_0`.

Example: RMB-10V

Let's say you are using an MEI RMB-10V and want to find the trigger for `XCVR_C`.

Look in `RMBMotorIoConfig` in `mei_rmb.h`. You will find that the appropriate value for `XCVR_C` is `RMBMotorIoConfigXCVR_C`. `RMBMotorIoConfigXCVR_C` is the third value in `RMBMotorIoConfig`. This means that the value to use in `MPICaptureSource` is `MPICaptureSourceMOTOR_IO_2` (the third `MPICaptureSourceMOTOR_IO` value).

A better way of making this conversion in your program is to add the `MPICaptureSourceMOTOR_IO_0` to the `nodeMotorIoConfig` value you want to use. In the above example, it would be `(MPICaptureSourceMOTOR_IO_0 + RMBMotorIoConfigXCVR_C)`.

Example: Trust TA800

To trigger off of hall A on a Trust TA800 node, you would use (MPICaptureSourceMOTOR_IO_0 + TA800MotorIoConfigHALL_A). Remember that you will need to look in *trust_ta800.h* (the node module) to find TA800MotorIoConfigHALL_A.

MPICaptureSourceMOTOR_IO_0	a capture trigger source is the 0 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_1	a capture trigger source is the 1 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_2	a capture trigger source is the 2 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_3	a capture trigger source is the 3 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_4	a capture trigger source is the 4 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_5	a capture trigger source is the 5 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_6	a capture trigger source is the 6 bit in the motor's configurable I/O.
MPICaptureSourceMOTOR_IO_7	a capture trigger source is the 7 bit in the motor's configurable I/O.
MPICaptureSourceHOME	a capture trigger source is the HOME input in the dedicated I/O input.
MPICaptureSourceINDEX	a capture trigger source is the encoder INDEX input in the dedicated I/O input.
MPICaptureSourceLIMIT_HW_NEG	a capture trigger source is the Hardware Negative Limit input in the dedicated I/O input.
MPICaptureSourceLIMIT_HW_POS	a capture trigger source is the Hardware Positive Limit input in the dedicated IO word. Please see MPIMotorInfoDedicatedIn .
MPICaptureSourceGLOBAL	a capture trigger source is the Global capture signal found on the node. Please see MPICaptureTriggerGlobal .

MPICaptureSourceINDEX_SECONDARY	A a capture trigger source is the index on the secondary encoder. If position based capture is selected with the feedback source being the secondary encoder, this is the only valid capture source.
MPICaptureSourceCOUNT	Total number of possible input sources for a capture.

See Also

[MPICaptureTrigger](#)

MPICaptureEdge

Definition

```
typedef enum MPICaptureEdge {  
    MPICaptureEdgeNONE,  
    MPICaptureEdgeRISING,  
    MPICaptureEdgeFALLING,  
    MPICaptureEdgeEITHER,  
} MPICaptureEdge;
```

Description

MPICaptureEdge is an enumeration of input trigger edge logic for a capture.

MPICaptureEdgeRISING	Triggers on a 0 to 1 transition.
MPICaptureEdgeFALLING	Triggers on a 1 to 0 transition.
MPICaptureEdgeEITHER	Triggers on either 0 to 1 or 1 to 0 transitions.

See Also

[MPICaptureTrigger](#)

MPICaptureTriggerGlobal

Definition

```
typedef struct MPICaptureTriggerGlobal {  
    MPI_BOOL    enabled;  
} MPICaptureTriggerGlobal;
```

Change History: Modified in the 03.03.00

Description

The **MPICaptureTriggerGlobal** structure specifies the global input trigger configuration for a capture.

enabled	Enables or disables the global input trigger. A value of TRUE enables the trigger, FALSE disables the trigger.
----------------	--

See Also

[MPICaptureConfig](#)

MPICaptureType

Definition

```
typedef enum {  
    MPICaptureTypePOSITION,  
    MPICaptureTypeTIME,  
} MPICaptureType;
```

Description

MPICaptureTypePOSITION	An actual position is captured by the Node from its feedback source.
MPICaptureTypeTIME	An internal timer is captured by the node and then a captured position is interpolated by the XMP firmware.

Remarks

Time-based capture will only work correctly if the speed of an axis is less than 344 million counts per second.

See Also

[MPICaptureConfig](#)

MPICaptureMessage / MEICaptureMessage

Definition: MPICaptureMessage

```
typedef enum {
    MPICaptureMessageMOTOR_INVALID,
    MPICaptureMessageCAPTURE_TYPE_INVALID,
    MPICaptureMessageCAPTURE_INVALID,
    MPICaptureMessageENCODER_INVALID,
} MPICaptureMessage;
```

Description

MPICaptureEdge is an enumeration of Capture error messages that can be returned by the MPI library.

MEICaptureMessageMOTOR_INVALID

[mpiCaptureConfigSet\(...\)](#) --> config.captureMotorNumber is not valid. It's either greater than maxMotors or == MPICaptureNOT.MAPPED.

MEICaptureMessageCAPTURE_TYPE_INVALID

[mpiCaptureConfigSet\(...\)](#) --> config.Type == MPICaptureNOT.MAPPED.

MPICaptureMessageCAPTURE_INVALID

The capture number is out of range. This message code is returned by [mpiCaptureCreate\(...\)](#) if the capture number is less than zero or greater than or equal to MEIXmpMaxCapturesPerMotor.

MPICaptureMessageENCODER_INVALID

The encoder index is out of range. This message code is returned by [mpiCaptureCreate\(...\)](#) if the encoder index is less than MPIMotorEncoderFIRST or greater than or equal to MPIMotorEncoderLAST. See [MPIMotorEncoder](#).

See Also

[mpiCaptureCreate](#) | [mpiControlConfigSet](#)

Definition: MEICaptureMessage


```
typedef enum {
    MEIOCaptureMessageINVALID_EDGE,
    MEIOCaptureMessageGLOBAL_CONFIG_ERR,
    MEIOCaptureMessageGLOBAL_ALREADY_ENABLED,
    MEIOCaptureMessageCAPTURE_NOT_ENABLED,
    MEIOCaptureMessageCAPTURE_STATE_INVALID,
    MEIOCaptureMessageNOT_MAPPED,
    MEIOCaptureMessageUNSUPPORTED_PRIMARY,
    MEIOCaptureMessageUNSUPPORTED_SECONDARY,
    MEIOCaptureMessageSECONDARY_INDEX_INVALID,
    MEIOCaptureMessageCAPTURE_ARMED,
} MEIOCaptureMessage;
```

Change History: Modified in the 03.03.00

Description

MEIOCaptureMessageINVALID_EDGE

The encoder edge trigger type is not valid. This message code is returned by [mpiCaptureConfigSet\(...\)](#) if the encoder capture edge type is not a member of the MPIOCaptureEdge enumeration.

MEIOCaptureMessageGLOBAL_CONFIG_ERR

The global trigger configuration is not valid. This message code is returned by [mpiCaptureConfigSet\(...\)](#) if the capture's trigger source is set to global and the capture's global trigger is enabled simultaneously. To correct this problem, either set the capture's trigger source to global or enable the capture's global trigger (not both).

MEIOCaptureMessage_GLOBAL_ALREADY_ENABLED

The global trigger is already enabled. This message code is returned by [mpiCaptureConfigSet\(...\)](#) if a global trigger is already enabled on another capture on the same node. Only one global trigger enable is allowed per node. To prevent this problem, do not enable a second global trigger on a single node.

MEIOCaptureMessageCAPTURE_NOT_ENABLED

This value is returned by [mpiCaptureCreate\(...\)](#) when the capture number specified is greater than the number of captures enabled in firmware. See [MPIOControlConfig](#).

MEIOCaptureMessageCAPTURE_STATE_INVALID

This value is returned by [mpiCaptureStatus\(...\)](#) when the communication between the controller and the capture logic on the node fails resulting in an invalid capture state. See [MPIOCaptureState](#).

MEIOCaptureMessageNOT_MAPPED

The capture object's hardware resource is not available. This message code is returned by [mpiCaptureCreate\(...\)](#) if the node hardware for the specified motor and encoder 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 and capture objects. A capture object cannot be created if there is no mapped hardware to support it. To correct this problem, verify that all expected nodes were found. Use [meiSynqNetInfo\(...\)](#) and [meiSqNodeInfo\(...\)](#) to determine the node topology and motor count per node. Check the node hardware power and network connections.

MEICaptureMessageUNSUPPORTED_PRIMARY

The capture hardware does not support the primary encoder. This message code is returned by [mpiCaptureCreate\(...\)](#) if the node hardware's primary encoder does not support the specified capture. To correct this problem, select a different motor, encoder, or capture number.

MEICaptureMessageUNSUPPORTED_SECONDARY

The capture hardware does not support the secondary encoder. This message code is returned by [mpiCaptureCreate\(...\)](#) if the node hardware's secondary encoder does not support the specified capture. To correct this problem, select a different motor, encoder, or capture number.

MEICaptureMessageSECONDARY_INDEX_INVALID

This message is returned from [MPICaptureConfigSet\(...\)](#) when the secondary encoder's index is specified as a trigger source in conjunction with other capture sources.

MEICaptureMessageCAPTURE_ARMED

See Also

[mpiCaptureCreate](#)

MPICaptureNOT_MAPPED

Definition

```
#define MPICaptureNOT_MAPPED (-1)
```

Description

MPICaptureEdge is an enumeration of input trigger edge logic for a capture.

Capture objects are associated with the controller and are not mapped to any hardware resources under default conditions. **MPICaptureNOT_MAPPED** will be assigned to:

```
long   captureMotorNumber;  
       long   feedbackMotorNumber;
```

when [mpiCaptureConfigGet](#) is called for the first time on a capture object. After a capture object has been used once, the resource mapping will remain in place until it is reassigned.

See Also