Adc Objects

Introduction

An **Adc** object manages a single analog-to-digital converter on an XMP motion controller. The XMP controller supports 8 channels (0 to 7) of 16 bit analog-to-digital conversions.

To configure the voltage range for an analog input, use the method **mpiAdcConfigSet**(MPIAdc adc, MPIAdcConfig *config, void *external), where *adc* is a handle to an MPIAdc object, and *config* is a pointer to a MPIAdcConfig structure. The MPIAdcConfig structure contains a "range" element, which specifies the analog input's voltage range. The XMP controller supports voltage ranges of 10.0V, 5.0V, 2.5V, or 1.25V volt ranges (all ranges are bipolar). Set the MPIAdcConfig range to one of the following values:

Bipolar Voltage Range (+/- V)
10.0
5.0
2.5
1.25

To use the ADC, first create an ADC object using **mpiAdcCreate**(MPIControl control, long number), where *control* is a handle to an MPIControl object and *number* is the analog-to-digital converter channel number.

The number of ADCs on the controller can be configured using the method **mpiControlConfigSet**(MPIControl control, MPIControlConfig *config, void *external). The MPIControlConfig structure contains an element called adcCount, which is used to specify the number of analog-to-digital converter channels. The default setting is for eight (8) axes.

After the ADC has been configured, to read a channel use the method **mpiAdc Input**(MPIAdc adc, unsigned long *input), where adc is a handle to an MPIAdc object, and input is a pointer to the digitally converted value.

Methods

Create, Delete, Validate Methods

mpiAdc**Create**	Create Adc object
mpiAdc**Delete**	Delete Adc object
mpiAdc**Validate**	Validate Adc object

Configuration and Information Methods

mpiAdc**ConfigGet**	Get Adc configuration
mpiAdc**ConfigSet**	Set Adc configuration
mpiAdc**FlashConfigGet**	Get Adc flash config
mpiAdc**FlashConfigSet**	Set Adc flash config

Action Methods

mpiAdc**Input**	Read input level

Memory Methods

mpiAdc**Memory**	Set address to Adc memory
mpiAdc**MemoryGet**	Copy Adc memory to application memory
mpiAdc**MemorySet**	Copy application memory to Adc memory

Relational Methods

mpiAdc**Control**	Return handle of Control object associated with Adc
mpiAdc**MotorMapGet**	Get the object map and write it into the structure pointed to by map
mpiAdc**Number**	Get index of Adc

Data Types

MPIAdc**Config** / MEIAdc**Config**
MPIAdc**Message**
MEIAdc**Mux**

mpiAdcCreate

Declaration

```
const MPIAdc mpiAdcCreate(MPIControl  control,
                          long        number)
```

Required Header

stdmpi.h

Description

AdcCreate creates an Adc object associated with the analog-to-digital converter identified by ***number*** located on motion controller ***control***. AdcCreate is the equivalent of a C++ constructor.

control	a handle to the motion controller.
number	the adc number

Return Values

MPIHandleVOID	if the object could not be created
handle	to an Adc object

See Also

mpiAdcDelete | mpiAdcValidate

mpiAdcDelete

Declaration `long mpiAdcDelete(MPIAdc adc)`

Required Header adc.h

Description **AdcDelete** deletes an Adc object and invalidates its handle (***adc***). *AdcDelete* is the equivalent of a C++ destructor.

Return Values	
MPIMessageOK	if *AdcDelete* successfully deletes the Adc object and invalidates its handle

See Also mpiAdcCreate | mpiAdcValidate

mpiAdcValidate

Declaration long **mpiAdcValidate**(MPIAdc **adc**)

Required Header stdmpi.h

Description **AdcValidate** validates the Adc object and its handle (***adc***).

Return Values	
MPIMessageOK	if Adc is a handle to a valid object.

See Also mpiAdcCreate | mpiAdcDelete

mpiAdcConfigGet

Declaration

```
long mpiAdcConfigGet(MPIAdc        adc,
                     MPIAdcConfig  *config,
                     void          *external)
```

Required Header

adc.h

Description

AdcConfigGet gets the configuration of an Adc object (***adc***) 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 configuration information in ***external*** is in addition to the configuration information in ***config***, i.e, the configuration information in ***config*** and in ***external*** is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL).

XMP Only	***external*** either points to a structure of type ***MEIAdcConfig{}*** or is NULL.

Return Values

MPIMessageOK	if *AdcConfigGet* successfully gets the configuration of an Adc object and writes it to the structure

See Also

mpiAdcConfigSet | MEIAdcConfig

mpiAdcConfigSet

Declaration

```
long mpiAdcConfigSet(MPIAdc        adc,
                     MPIAdcConfig  *config,
                     void          *external)
```

Required Header

stdmpi.h

Description

AdcConfigSet sets the configuration of an Adc object (***adc***) using 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 configuration information in ***external*** is in addition to the configuration information in ***config***, i.e, the configuration information in ***config*** and in ***external*** is not the same information. Note that ***config*** or ***external*** can be NULL (but not both NULL).

XMP Only ***external*** either points to a structure of type ***MEIAdcConfig{}*** or is NULL.

Return Values

MPIMessageOK	if *AdcConfigSet* successfully sets the configuration of the Adc object using data from the structure

See Also

mpiAdcConfigGet | MEIAdcConfig

mpiAdcFlashConfigGet

Declaration

```
long mpiAdcFlashConfigGet(MPIAdc          adc,
                          void            *flash,
                          MPIAdcConfig    *config,
                          void            *external
```

Required Header

stdmpi.h

Description

AdcFlashConfigGet gets the flash configuration of an Adc object (***adc***) and writes it into the structure pointed to by ***config***, and also writes it in the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

XMP Only

external either points to a structure of type **MEIAdcConfig{}** or is NULL. **flash** is either an MEIFlash handle or MPIHandleVOID. If **flash** is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

Return Values

MPIMessageOK	if *AdcFlashConfigGet* successfully gets the Adc's flash configuration and writes it into the structure(s)

See Also

mpiAdcFlashConfigSet | MEIAdcConfig | MEIFlash

mpiAdcFlashConfigSet

Declaration

```
long mpiAdcFlashConfigSet(MPIAdc          adc,
                          void            *flash,
                          MPIAdcConfig    *config,
                          void            *external)
```

Required Header

stdmpi.h

Description

AdcFlashConfigSet sets the flash configuration of an Adc object (***adc***) using the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

XMP Only	**external** either points to a structure of type **MEIAdcConfig{}** or is NULL. **flash** is either an MEIFlash handle or MPIHandleVOID. If **flash** is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

Return Values

MPIMessageOK	if *AdcFlashConfigSet* successfully sets the Adc's flash configuration using data from the structure(s)

See Also

MEIFlash | mpiAdcFlashConfigGet

mpiAdcInput

Declaration

```
long mpiAdcInput(MPIAdc          adc,
                 unsigned long   *input)
```

Required Header

stdmpi.h

Description

AdcInput gets the value of an Adc's (***adc***) input and writes it into the location pointed to by ***input***.

Return Values

MPIMessageOK	if *AdcInput* successfully gets the value of the Adc's input and writes it to the location

See Also

mpiAdcMemory

Declaration

```
long mpiAdcMemory(MPIAdc     adc,
                  void       **memory)
```

Required Header

stdmpi.h

Description

AdcMemory writes an address [used to access an Adc's memory (***adc***)] to the contents of ***memory***. This address, or an address calculated from it, can be passed as the src argument to mpiAdcMemoryGet(...) and the ***dst*** argument to mpiAdcMemorySet(...).

Return Values

MPIMessageOK	if *AdcMemory* successfully writes the Adc's address to the contents of ***memory***

See Also

mpiAdcMemoryGet | mpiAdcMemorySet

mpiAdcMemoryGet

Declaration

```
long mpiAdcMemoryGet(MPIAdc    adc,
                     void      *dst,
                     void      *src,
                     long      count)
```

Required Header

stdmpi.h

Description

AdcMemoryGet copies ***count*** bytes of an Adc's memory (***adc***, starting at address ***src***) and to application memory (starting at address ***dst***).

Return Values

MPIMessageOK	if *AdcMemoryGet* successfully copies Adc memory to application memory

See Also

mpiAdcMemorySet | mpiAdcMemory

mpiAdcMemorySet

Declaration

```
long mpiAdcMemorySet(MPIAdc   adc,
                     void     *dst,
                     void     *src,
                     long     count)
```

Required Header

stdmpi.h

Description

AdcMemorySet copies ***count*** bytes of application memory (starting at address ***src***) to an Adc's memory (***adc***, starting at address ***dst***).

Return Values

MPIMessageOK	if *AdcMemorySet* successfully copies application memory to Adc memory

See Also

mpiAdcMemoryGet | mpiAdcMemory

mpiAdcControl

Declaration `const MPIControl mpiAdcControl(MPIAdc adc)`

Required Header stdmpi.h

Description **AdcControl** returns a handle to the motion controller (Control) with which an Adc (***adc***) is associated.

adc	a handle to the Adc object

Return Values	
MPIHandleVOID	if **adc** is invalid
MPIControl	handle to a Control object

See Also mpiAdcCreate | mpiControlCreate

mpiAdcMotorMapGet

Declaration

```
long mpiAdcMotorMapGet(MPIAdc          adc,
                       MPIObjectMap    *map)
```

Required Header

stdmpi.h

Description

AdcMotorMapGet gets the object map [of the Motors associated with an Adc (***adc***)] and writes it into the structure pointed to by ***map***.

Return Values

MPIMessageOK	if *AdcMotorMapGet* successfully gets the object map of the Motors associated with an Adc and writes it into the structure

See Also

mpiAdcNumber

Declaration

```
long mpiAdcNumber(MPIAdc    adc,
                  long      *number)
```

Required Header

stdmpi.h

Description

AdcNumber writes the index of an Adc object (***adc***, on the motion controller that the Adc object is associated with) to the contents of ***number***.

Return Values

MPIMessageOK	if *AdcNumber* successfully writes the Adc's index to the contents of ***number***

See Also

MPIAdcConfig / MEIAdcConfig

MPIAdcConfig

```
typedef struct MPIAdcConfig {
    double      range;  /* +/- Voltage */
} MPIAdcConfig;
```

Description

range	The voltage range that the ADC can be configured for. All values are measured in volts. Valid values are 2.5, 5.0, and 10.0

MEIAdcConfig

```
typedef struct MEIAdcConfig {
    MEIAdcMux   mux;
} MEIAdcConfig;
```

Description

mux	is used to configure what signal an MPIAdc object will look at.

See Also

MPIAdc

MPIAdcMessage

MPIAdcMessage

```
typedef enum {
                MPIAdcMessageADC_INVALID,
        } MPIAdcMessage;
```

Description

MPIAdcMessageADC_INVALID	
Meaning	The MPIAdc handle passed to an MPIAdc method is invalid.
Possible Causes	Either the handle was never initialized or the mpiAdcCreate method failed.
Recommendations	Use mpiAdcValidate after mpiAdcCreate to see if the returned handle is valid.

See Also

mpiAdcValidate | mpiAdcCreate

MEIAdcMux

MEIAdcMux

```
typedef enum {
   MEIAdcMuxDAC_CAL,
   MEIAdcMuxPLUS10V,
   MEIAdcMux0V,
   MEIAdcMuxMINUS10V,
   MEIAdcMuxANALOG_IN_0,
   MEIAdcMuxANALOG_IN_1,
   MEIAdcMuxANALOG_IN_2,
   MEIAdcMuxANALOG_IN_3,
   MEIAdcMuxANALOG_IN_4,
   MEIAdcMuxANALOG_IN_5,
   MEIAdcMuxANALOG_IN_6,
   MEIAdcMuxANALOG_IN_7,
} MEIAdcMux;
```

Description

AdcMux is an enumeration used by MEIAdcConfig.mux. It configures what signal a particular MPIAdc object will look at. The following table lists what signal each MEIAdcMux enumeration stands for:

MEIAdcMuxDAC_CAL	Used to calibrate an ADC object from a DAC object. Used by MEI for calibration and testing.
MEIAdcMuxPLUS10V	MEIAdcMuxPLUS10V +10V signal. Used by MEI for calibration and testing.
MEIAdcMux0V	0V signal. Used by MEI for calibration and testing.
MEIAdcMuxMINUS10V	-10V signal. Used by MEI for calibration and testing.
MEIAdcMuxANALOG_IN_0	Analog input 0
MEIAdcMuxANALOG_IN_1	Analog input 1
MEIAdcMuxANALOG_IN_2	Analog input 2
MEIAdcMuxANALOG_IN_3	Analog input 3
MEIAdcMuxANALOG_IN_4	Analog input 4
MEIAdcMuxANALOG_IN_5	Analog input 5
MEIAdcMuxANALOG_IN_6	Analog input 6
MEIAdcMuxANALOG_IN_7	Analog input 7

Note

MEIAdcMuxDAC_CAL	These are all specific to XMP Revision 1 boards. The ADC input lines will be floating if these values are used to configure an MPIAdc object.
MEIAdcMuxPLUS10V	
MEIAdcMux0V	
MEIAdcMuxMINUS10V	

See Also MEIAdcConfig