ADS1x15

Arduino library for the TI **ADS1115** 16bit and **ADS1015** 12bit I2C Analog to Digital Converters

Hierarchical Index

Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

File Index

File List

Here is a list of all files with brief descriptions:

Class Documentation

ADS1015 Class Reference

Interface class for the **ADS1015** analog to digital converter.
`#include <ADS1x15.h>`
Inherits **ADS1x15**.

Public Member Functions

- **ADS1015** ()
- void **setDataRate** (**ADS1015_DR_t**)
  Set the conversion rate in samples per second.
- uint8_t **getADCbits** ()
  Get the number of bits of the current ADC.
- uint16_t **getFullScaleBits** ()
  Get the full scale binary output for the chip.

Additional Inherited Members

Detailed Description

Interface class for the **ADS1015** analog to digital converter.

Constructor & Destructor Documentation

ADS1015::ADS1015 ()[inline]

Member Function Documentation

uint8_t ADS1015::getADCbits ()[inline], [virtual]

Get the number of bits of the current ADC.

Returns:
Number of bits
Reimplemented from **ADS1x15** (*p.8*).

uint16_t ADS1015::getFullScaleBits () [inline], [virtual]

Get the full scale binary output for the chip.

Returns:

Full scale output

Reimplemented from **ADS1x15** (*p.8*).

void ADS1015::setDataRate (ADS1015_DR_t *dataRate*)

Set the conversion rate in samples per second.

Parameters:

dataRate	One of the rate settings from ADS1015_DR_t

The documentation for this class was generated from the following files:

- src/**ADS1x15.h**
- src/**ADS1x15.cpp**

ADS1115 Class Reference

Interface class for the **ADS1115** analog to digital converter.
`#include <ADS1x15.h>`
Inherits **ADS1x15**.

Public Member Functions

- **ADS1115** ()
- void **setDataRate** (**ADS1115_DR_t**)
  Set the conversion rate in samples per second.
- uint8_t **getADCbits** ()
  Get the number of bits of the current ADC.
- uint16_t **getFullScaleBits** ()
  Get the full scale binary output for the chip.

Additional Inherited Members

Detailed Description

Interface class for the **ADS1115** analog to digital converter.

Constructor & Destructor Documentation

ADS1115::ADS1115 ()[inline]

Member Function Documentation

uint8_t ADS1115::getADCbits ()[inline], [virtual]

Get the number of bits of the current ADC.

Returns:
Number of bits
Reimplemented from **ADS1x15** (*p.8*).

uint16_t ADS1115::getFullScaleBits () [inline], [virtual]

Get the full scale binary output for the chip.

Returns:
Full scale output
Reimplemented from **ADS1x15** (*p.8*).

void ADS1115::setDataRate (ADS1115_DR_t *dataRate*)

Set the conversion rate in samples per second.

Parameters:

dataRate	One of the rate settings from ADS1115_DR_t

The documentation for this class was generated from the following files:

- src/**ADS1x15.h**
- src/**ADS1x15.cpp**

ADS1x15 Class Reference

Foundation class for the **ADS1015** and **ADS1115** ADCs.
`#include <ADS1x15.h>`
Inherits **wireUtil< ADS1x15_Register_t, uint16_t >**.
Inherited by **ADS1015**, and **ADS1115**.

Public Member Functions

- **ADS1x15** ()
- void **begin** ()
- uint8_t **addressIndex** (uint8_t a)
- void **setCalibration** (float)
  Set the calibration factor for calculating the voltage or current input.
- void **setCalibration** (float, float)
  Calculate the calibration factor for calculating the voltage or current input.
- void **setGain** (**ADS1x15_GAIN_t**)
  Set the gain value for the programmable gain amplifier.
- float **getFullScaleV** ()
  Get the current full scale value in V.
- void **setComparatorMode** (**ADS1x15_COMP_MODE_t**)
  Set the mode of the comparator.
- void **setComparatorPolarity** (**ADS1x15_COMP_POL_t**)
  Set the polarity of the comparator.
- void **setComparatorLatch** (**ADS1x15_COMP_LAT_t**)
  Set the latching mode of the comparator.
- int16_t **analogRead** (**ADS1x15_MUX_t**)
  Read an analog value.
- uint16_t **analogRead** (uint8_t)
  Read an analog value.
- float **analogReadVoltage** (uint8_t)
  Read an input and calculate the voltage based on the current gain settings.
- float **analogReadCurrent** (uint8_t, float=100.0)
  [brief description]
- float **analogRead420** (uint8_t, float=100.0)
  Read the output from a 4-20mA device in %.
- float **getCalibration** ()
- virtual uint8_t **getADCbits** ()
- virtual uint16_t **getFullScaleBits** ()

Protected Member Functions

- virtual uint16_t **shiftConversion** (uint16_t c)

Protected Attributes

- uint16_t **configRegister**
- **ADS1x15_GAIN_t currentGain**
- uint32_t **conversionDelay**
- float **calibration**

Additional Inherited Members

Detailed Description

Foundation class for the **ADS1015** and **ADS1115** ADCs.

Constructor & Destructor Documentation

ADS1x15::ADS1x15 ()[inline]

Member Function Documentation

uint8_t ADS1x15::addressIndex (uint8_t *a*)[inline]

int16_t ADS1x15::analogRead (ADS1x15_MUX_t *mux*)

Read an analog value.

Parameters:

m	The configuration of the MUX

Returns:

The converted value

uint16_t ADS1x15::analogRead (uint8_t *ch*)

Read an analog value.

Parameters:

ch	The input channel to read

Returns:

The converted value

float ADS1x15::analogRead420 (uint8_t *ch*, float *r* = `100.0`)

Read the output from a 4-20mA device in %.

Parameters:

ch	The input channel to read
r	Burden resistor value in ohms

Returns:

The converted value in %

float ADS1x15::analogReadCurrent (uint8_t *ch*, float *r* = `100.0`)

[brief description]

[long description]

Parameters:

ch	The input channel to read
r	Burden resistor value in ohms

Returns:

The converted value in mA

float ADS1x15::analogReadVoltage (uint8_t *ch*)

Read an input and calculate the voltage based on the current gain settings.

Parameters:

ch	The input channel to read

Returns:

The converted value in V

void ADS1x15::begin () `[inline]`, `[virtual]`

Reimplemented from **wireUtil< ADS1x15_Register_t, uint16_t >** (*p.12*).

virtual uint8_t ADS1x15::getADCbits () `[virtual]`

Reimplemented in **ADS1015** (*p.2*), and **ADS1115** (*p.4*).

float ADS1x15::getCalibration () `[inline]`

virtual uint16_t ADS1x15::getFullScaleBits () `[virtual]`

Reimplemented in **ADS1015** (*p.3*), and **ADS1115** (*p.5*).

float ADS1x15::getFullScaleV ()

Get the current full scale value in V.

Returns:

Voltage based on the current gain and calibration factor

void ADS1x15::setCalibration (float *calibration*)

Set the calibration factor for calculating the voltage or current input.

Parameters:

calibration	Correction factor

void ADS1x15::setCalibration (float *r1*, float *r2*)

Calculate the calibration factor for calculating the voltage or current input.

Parameters:

r1	First resistor in the resistor divider
r2	Second resistor in the resistor divider

void ADS1x15::setComparatorLatch (ADS1x15_COMP_LAT_t *compCfg*)

Set the latching mode of the comparator.

Parameters:

compCfg	Configuration to set

void ADS1x15::setComparatorMode (ADS1x15_COMP_MODE_t *compCfg*)

Set the mode of the comparator.

Parameters:

compCfg	Configuration to set

void ADS1x15::setComparatorPolarity (ADS1x15_COMP_POL_t *compCfg*)

Set the polarity of the comparator.

Parameters:

compCfg	Configuration to set

void ADS1x15::setGain (ADS1x15_GAIN_t *currentGain*)

Set the gain value for the programmable gain amplifier.

Parameters:

currentGain	Gain value from ADS1x15_GAIN_t

virtual uint16_t ADS1x15::shiftConversion (uint16_t *c*)`[inline]`, `[protected]`, `[virtual]`

Member Data Documentation

float ADS1x15::calibration`[protected]`

uint16_t ADS1x15::configRegister`[protected]`

uint32_t ADS1x15::conversionDelay`[protected]`

ADS1x15_GAIN_t ADS1x15::currentGain`[protected]`

The documentation for this class was generated from the following files:

- src/**ADS1x15.h**
- src/**ADS1x15.cpp**

wireUtil< REGTYPE, DATATYPE > Class Template Reference

Utility base class for reading and writing registers on i2c devices.
`#include <wireUtil.h>`

Public Member Functions

- void **attachTimeoutHandler** (void(*timeOutHandler)(void))
  Attach a function to be called on a read timeout.
- void **attachErrorHandler** (void(*errorHandler)(uint8_t))
  Attach a function to be called on a write NACK.
- bool **getTimeoutFlag** ()
  Safe method to read the state of the timeout flag.
- virtual void **begin** ()
- virtual void **begin** (uint8_t)
  Initialize the chip at a specific address.
- bool **writeRegister** (REGTYPE, DATATYPE)
  Write a single register on an i2c device.
- bool **writeRegisters** (REGTYPE, DATATYPE *, uint8_t)
  Write to a sequence of registers on an i2c device.
- DATATYPE **readRegister** (REGTYPE)
  Read a single register from an i2c device.
- bool **readRegisters** (REGTYPE, DATATYPE *, uint8_t)
  Read a number of sequential registers from an i2c device.
- bool **setRegisterBit** (REGTYPE, uint8_t, bool)
  Read modify write a bit on a register.

Public Attributes

- unsigned long **timeoutTime**
  Amount of time to wait for a successful read.
- bool **timeoutFlag**
  Set to true if there is a timeout event, reset on the next read.

Protected Attributes

- uint8_t **address**
  Hardware address of the device.

Detailed Description

template<typename REGTYPE, typename DATATYPE = uint8_t>

class wireUtil< REGTYPE, DATATYPE >

Utility base class for reading and writing registers on i2c devices.

Template Parameters:

REGTYPE	An initialized enum type that lists the valid registers for the device
DATATYPE	= uint8_t Data type (register size) supports uint8_t, uint16_t, uint32_t

---

Member Function Documentation

template<typename REGTYPE, typename DATATYPE = uint8_t> void wireUtil< REGTYPE, DATATYPE >::attachErrorHandler (void(*)(uint8_t) *errorHandler*)[inline]

Attach a function to be called on a write NACK.

Parameters:

errorHandler	Pointer to a 'void f(uint8_t)' function. This will be passed the Wire status.

template<typename REGTYPE, typename DATATYPE = uint8_t> void wireUtil< REGTYPE, DATATYPE >::attachTimeoutHandler (void(*)(void) *timeOutHandler*)[inline]

Attach a function to be called on a read timeout.

Parameters:

timeOutHandler	Pointer to a 'void f(void)' function

template<typename REGTYPE, typename DATATYPE = uint8_t> virtual void wireUtil< REGTYPE, DATATYPE >::begin ()[virtual]

Reimplemented in **ADS1x15** (*p.8*).

template<typename REGTYPE , typename DATATYPE > void wireUtil< REGTYPE, DATATYPE >::begin (uint8_t *address*)[virtual]

Initialize the chip at a specific address.

Parameters:

address	Address of the chip

template<typename REGTYPE, typename DATATYPE = uint8_t> bool wireUtil< REGTYPE, DATATYPE >::getTimeoutFlag ()[inline]

Safe method to read the state of the timeout flag.

Returns:

State of the timeout flag

template<typename REGTYPE, typename DATATYPE > DATATYPE wireUtil< REGTYPE, DATATYPE >::readRegister (REGTYPE *reg*)

Read a single register from an i2c device.

Parameters:

reg	Register address (from a device specific enum)

Returns:

Data from the device register, 0 if there is a timeout

template<typename REGTYPE, typename DATATYPE> bool wireUtil< REGTYPE, DATATYPE >::readRegisters (REGTYPE *reg*, DATATYPE * *buffer*, uint8_t *len*)

Read a number of sequential registers from an i2c device.

Parameters:

reg	First register address (from a device specific enum)
buffer	Array to contain the data read
len	Number of bytes to read

Returns:

true on success, false on timeout

template<typename REGTYPE, typename DATATYPE > bool wireUtil< REGTYPE, DATATYPE >::setRegisterBit (REGTYPE *reg*, uint8_t *bit*, bool *state*)

Read modify write a bit on a register.

Parameters:

reg	register to modify
bit	index of the bit to set
state	state of the bit to set

Returns:

true on success

template<typename REGTYPE, typename DATATYPE> bool wireUtil< REGTYPE, DATATYPE >::writeRegister (REGTYPE *reg*, DATATYPE *data*)

Write a single register on an i2c device.

Parameters:

reg	Register address (from a device specific enum)
data	Data to be written to the device

Returns:

true on success, false if NACK

template<typename REGTYPE, typename DATATYPE> bool wireUtil< REGTYPE, DATATYPE >::writeRegisters (REGTYPE *reg*, DATATYPE * *buffer*, uint8_t *len*)

Write to a sequence of registers on an i2c device.

Parameters:

reg	First register address (from a device specific enum)
buffer	Array containing the data to be written
len	Number of bytes in the array

Returns:

true on success, false if NACK

Member Data Documentation

template<typename REGTYPE, typename DATATYPE = uint8_t> uint8_t wireUtil< REGTYPE, DATATYPE >::address[protected]

Hardware address of the device.

template<typename REGTYPE, typename DATATYPE = uint8_t> bool wireUtil< REGTYPE, DATATYPE >::timeoutFlag

Set to true if there is a timeout event, reset on the next read.

template<typename REGTYPE, typename DATATYPE = uint8_t> unsigned long wireUtil< REGTYPE, DATATYPE >::timeoutTime

Amount of time to wait for a successful read.

The documentation for this class was generated from the following file:

- src/utility/**wireUtil.h**

src/ADS1x15.cpp File Reference

```
#include "ADS1x15.h"
```

src/ADS1x15.h File Reference

```
#include <Arduino.h>
#include <Wire.h>
#include "utility/wireUtil.h"
```

Classes

- class **ADS1x15**
- *Foundation class for the **ADS1015** and **ADS1115** ADCs.* class **ADS1115**
- *Interface class for the **ADS1115** analog to digital converter.* class **ADS1015**

Interface class for the ADS1015 analog to digital converter. Typedefs

- typedef **ADS1x15_GAIN_t ADS1015_GAIN_t**
- typedef **ADS1x15_GAIN_t ADS1115_GAIN_t**

Enumerations

- enum **ADS1x15_Register_t** { **CONVERSION_REG** = 0x00, **CONFIG_REG** = 0x01, **LOW_THRESH_REG** = 0x02, **HI_THRESH_REG** = 0x03 }
- enum **ADS1x15_MUX_t** { **DIF01** = (0x0 << 12), **DIF03** = (0x1 << 12), **DIF13** = (0x2 << 12), **DIF23** = (0x3 << 12), **SE0** = (0x4 << 12), **SE1** = (0x5 << 12), **SE2** = (0x6 << 12), **SE3** = (0x7 << 12) }
- enum **ADS1x15_GAIN_t** { **GAIN_23** = (0x0 << 9), **GAIN_1** = (0x1 << 9), **GAIN_2** = (0x2 << 9), **GAIN_4** = (0x3 << 9), **GAIN_8** = (0x4 << 9), **GAIN_16** = (0x5 << 9) }
- enum **ADS1x15_MODE_t** { **CONTINUOUS_CONV** = 0x0 << 8, **SINGLE_SHOT** = 0x1 << 8 }
- enum **ADS1115_DR_t** { **ADS1115_DR_8** = (0x0 << 5), **ADS1115_DR_16** = (0x1 << 5), **ADS1115_DR_32** = (0x2 << 5), **ADS1115_DR_64** = (0x3 << 5), **ADS1115_DR_128** = (0x4 << 5), **ADS1115_DR_250** = (0x5 << 5), **ADS1115_DR_475** = (0x6 << 5), **ADS1115_DR_860** = (0x7 << 5) }
- enum **ADS1015_DR_t** { **ADS1015_DR_128** = (0x0 << 5), **ADS1015_DR_250** = (0x1 << 5), **ADS1015_DR_490** = (0x2 << 5), **ADS1015_DR_920** = (0x3 << 5), **ADS1015_DR_1600** = (0x4 << 5), **ADS1015_DR_2400** = (0x5 << 5), **ADS1015_DR_3300** = (0x6 << 5) }
- enum **ADS1x15_COMP_MODE_t** { **STANDARD_COMP** = 0x0 << 4, **WINDOW_COMP** = 0x1 << 4 }
- enum **ADS1x15_COMP_POL_t** { **ACTIVE_LOW** = 0x0 << 3, **ACTIVE_HIGH** = 0x1 << 3 }
- enum **ADS1x15_COMP_LAT_t** { **NONLATCHING_COMP** = 0x0 << 2, **LATCHING_COMP** = 0x1 << 2 }
- enum **ADS1x15_QUE_t** { **QUE_ONE** = 0x0, **QUE_TWO** = 0x1, **QUE_FOUR** = 0x2, **QUE_DISABLE** = 0x3 }

Typedef Documentation

typedef ADS1x15_GAIN_t ADS1015_GAIN_t

typedef ADS1x15_GAIN_t ADS1115_GAIN_t

Enumeration Type Documentation

enum ADS1015_DR_t

Enumerator

ADS1015_DR_128
ADS1015_DR_250
ADS1015_DR_490
ADS1015_DR_920
ADS1015_DR_1600
ADS1015_DR_2400
ADS1015_DR_3300

enum ADS1115_DR_t

Enumerator

ADS1115_DR_8
ADS1115_DR_16
ADS1115_DR_32
ADS1115_DR_64
ADS1115_DR_128
ADS1115_DR_250
ADS1115_DR_475
ADS1115_DR_860

enum ADS1x15_COMP_LAT_t

Enumerator

NONLATCHING_COMP
LATCHING_COMP

enum ADS1x15_COMP_MODE_t

Enumerator

STANDARD_COMP
WINDOW_COMP

enum ADS1x15_COMP_POL_t

Enumerator

ACTIVE_LOW
ACTIVE_HIGH

enum ADS1x15_GAIN_t

Enumerator

GAIN_23
GAIN_1
GAIN_2
GAIN_4
GAIN_8
GAIN_16

enum ADS1x15_MODE_t

Enumerator

CONTINUOUS_CONV
SINGLE_SHOT

enum ADS1x15_MUX_t

Enumerator

DIF01
DIF03
DIF13
DIF23
SE0
SE1
SE2
SE3

enum ADS1x15_QUE_t

Enumerator

QUE_ONE
QUE_TWO
QUE_FOUR
QUE_DISABLE

enum ADS1x15_Register_t

Enumerator

CONVERSION_REG
CONFIG_REG
LOW_THRESH_REG
HI_THRESH_REG

src/utility/wireUtil.h File Reference

Utility base class for reading and writing registers on i2c devices.

```
#include <Arduino.h>
#include <Wire.h>
```

Classes

- class **wireUtil< REGTYPE, DATATYPE >**

Utility base class for reading and writing registers on i2c devices.

Detailed Description

Utility base class for reading and writing registers on i2c devices.

Author:

Keegan Morrow

Version:

1.1.2

Index

INDEX