Data acquisition with the ADS1115 on the raspberry PI

Generated by Doxygen 1.9.8

Chapter 1

rpi_ads1115

Raspberry PI C++ library for the ADS1115

github: https://github.com/berndporr/rpi_ads1115

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 ADS1115rpi Class Reference

This class reads data from the ADS1115 in the background (separate thread) and calls a callback function whenever data is available.

`#include <ads1115rpi.h>`

Public Types

- using ADSCallbackInterface = std::function< void(float)>

  Callback function type when a new sample is available.

Public Member Functions

- ~**ADS1115rpi** ()

  Destructor which makes sure the data acquisition stops on exit.
- void **registerCallback** (ADSCallbackInterface ci)
- void setChannel (ADS1115settings::Input channel)

  Selects a different channel at the multiplexer while running.
- void start (ADS1115settings settings=ADS1115settings())

  Starts the data acquisition in the background and the callback is called with new samples.
- ADS1115settings **getADS1115settings** () const

  Returns the current settings.
- void **stop** ()

  Stops the data acquistion.

4.1.1 Detailed Description

This class reads data from the ADS1115 in the background (separate thread) and calls a callback function whenever data is available.

4.1.2 Member Typedef Documentation

4.1.2.1 ADSCallbackInterface

```
using ADS1115rpi::ADSCallbackInterface = std::function<void(float)>
```

Callback function type when a new sample is available.

Value is in volt.

4.1.3 Member Function Documentation

4.1.3.1 setChannel()

```
void ADS1115rpi::setChannel (
            ADS1115settings::Input channel )
```

Selects a different channel at the multiplexer while running.

Call this in the callback handler hasSample() to cycle through different channels.

Parameters

channel	Sets the channel from A0..A3.

4.1.3.2 start()

```
void ADS1115rpi::start (
            ADS1115settings settings = ADS1115settings() )
```

Starts the data acquisition in the background and the callback is called with new samples.

Parameters

settings	A struct with the settings.

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

- ads1115rpi.h

4.2 ADS1115settings Struct Reference

ADS1115 initial settings when starting the device.

```
#include <ads1115rpi.h>
```

Public Types

- enum SamplingRates {
  FS8HZ = 0 , **FS16HZ** = 1 , **FS32HZ** = 2 , **FS64HZ** = 3 ,
  FS128HZ = 4 , **FS250HZ** = 5 , **FS475HZ** = 6 , **FS860HZ** = 7 }

  Sampling rates.
- enum PGA { **FSR2_048** = 2 , **FSR1_024** = 3 , **FSR0_512** = 4 , **FSR0_256** = 5 }

  Full scale range: 2.048V, 1.024V, 0.512V or 0.256V.
- enum Input { **AIN0** = 0 , **AIN1** = 1 , **AIN2** = 2 , **AIN3** = 3 }

  Channel indices.

Public Member Functions

- unsigned **getSamplingRate** () const

  Get the sampling rate in Hz.

Public Attributes

- int **i2c_bus** = 1

  I2C bus used (99% always set to one)
- uint8_t **address** = DEFAULT_ADS1115_ADDRESS

  I2C address of the ads1115.
- SamplingRates **samplingRate** = FS8HZ

  Sampling rate requested.
- PGA **pgaGain** = FSR2_048

  Requested full scale range.
- Input **channel** = AIN0

  Requested input channel (AIN0..AIN3)
- int **drdy_chip** = 0

  GPIO Chip number which receives the Data Ready signal.
- int **drdy_gpio** = DEFAULT_ALERT_RDY_TO_GPIO

  GPIO pin connected to ALERT/RDY.

4.2.1 Detailed Description

ADS1115 initial settings when starting the device.

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

- ads1115rpi.h

Chapter 5

File Documentation

5.1 ads1115rpi.h

```
00001 #ifndef __ADS1115RPI_H
00002 #define __ADS1115RPI_H
00003
00004 /*
00005  * ADS1115 class to read data at a given sampling rate
00006  *
00007  * Copyright (c) 2007  MontaVista Software, Inc.
00008  * Copyright (c) 2007  Anton Vorontsov <avorontsov@ru.mvista.com>
00009  * Copyright (c) 2013-2025  Bernd Porr <mail@berndporr.me.uk>
00010  *
00011  * This program is free software; you can redistribute it and/or modify
00012  * it under the terms of the GNU General Public License as published by
00013  * the Free Software Foundation; either version 2 of the License.
00014  *
00015  */
00016 #include <stdint.h>
00017 #include <unistd.h>
00018 #include <stdio.h>
00019 #include <stdlib.h>
00020 #include <assert.h>
00021 #include <linux/i2c-dev.h>
00022 #include <thread>
00023 #include <gpiod.hpp>
00024 #include <functional>
00025
00026 // enable debug messages and error messages to stderr
00027 #ifndef NDEBUG
00028 #define DEBUG
00029 #endif
00030
00031 static const char could_not_open_i2c[] = "Could not open I2C.\n";
00032
00033 #define ISR_TIMEOUT_MS 500
00034
00035 // default address if ADDR is pulled to GND
00036 #define DEFAULT_ADS1115_ADDRESS 0x48
00037
00038 // default GPIO pin for the ALRT/DRY signal
00039 #define DEFAULT_ALERT_RDY_TO_GPIO 17
00040
00041
00042
00043
00047 struct ADS1115settings {
00048
00052     int i2c_bus = 1;
00053
00057     uint8_t address = DEFAULT_ADS1115_ADDRESS;
00058
00062     enum SamplingRates {
00063     FS8HZ   = 0,
00064     FS16HZ  = 1,
00065     FS32HZ  = 2,
00066     FS64HZ  = 3,
00067     FS128HZ = 4,
00068     FS250HZ = 5,
00069     FS475HZ = 6,
00070     FS860HZ = 7
```

```
00071     };
00072
00076     inline unsigned getSamplingRate() const {
00077     const unsigned SamplingRateEnum2Value[8] =
00078         {8,16,32,64,128,250,475,860};
00079     return SamplingRateEnum2Value[samplingRate];
00080     }
00081
00085     SamplingRates samplingRate = FS8HZ;
00086
00090     enum PGA {
00091     FSR2_048 = 2,
00092     FSR1_024 = 3,
00093     FSR0_512 = 4,
00094     FSR0_256 = 5
00095     };
00096
00100     PGA pgaGain = FSR2_048;
00101
00105     enum Input {
00106     AIN0 = 0,
00107     AIN1 = 1,
00108     AIN2 = 2,
00109     AIN3 = 3
00110     };
00111
00115     Input channel = AIN0;
00116
00120     int drdy_chip = 0;
00121
00125     int drdy_gpio = DEFAULT_ALERT_RDY_TO_GPIO;
00126 };
00127
00128
00133 class ADS1115rpi {
00134
00135 public:
00140     ~ADS1115rpi() {
00141     stop();
00142     }
00143
00147     using ADSCallbackInterface = std::function<void(float)>;
00148
00149     void registerCallback(ADSCallbackInterface ci) {
00150     adsCallbackInterface = ci;
00151     }
00152
00160     void setChannel(ADS1115settings::Input channel);
00161
00167     void start(ADS1115settings settings = ADS1115settings() );
00168
00172     ADS1115settings getADS1115settings() const {
00173     return ads1115settings;
00174     }
00175
00179     void stop();
00180
00181 private:
00182     ADS1115settings ads1115settings;
00183
00184     void dataReady();
00185
00186     void worker();
00187
00188     void i2c_writeWord(uint8_t reg, unsigned data);
00189     unsigned i2c_readWord(uint8_t reg);
00190     int i2c_readConversion();
00191
00192     const uint8_t reg_config = 1;
00193     const uint8_t reg_lo_thres = 2;
00194     const uint8_t reg_hi_thres = 3;
00195
00196     float fullScaleVoltage() {
00197     switch (ads1115settings.pgaGain) {
00198     case ADS1115settings::FSR2_048:
00199         return 2.048f;
00200     case ADS1115settings::FSR1_024:
00201         return 1.024f;
00202     case ADS1115settings::FSR0_512:
00203         return 0.512f;
00204     case ADS1115settings::FSR0_256:
00205         return 0.256f;
00206     }
00207     assert(1 == 0);
00208     return 0;
00209     }
00210
```

```
00211     std::shared_ptr<gpiod::chip> chip;
00212     std::shared_ptr<gpiod::line_request> request;
00213 
00214     std::thread thr;
00215 
00216     int fd_i2c = -1;
00217 
00218     bool running = false;
00219 
00220     ADSCallbackInterface adsCallbackInterface;
00221 };
00222 
00223 
00224 #endif
```

Index