libnewi2c

Version 1.0.0 – 16 June 2016
Kit Bishop

Document History		
Version	**Date**	**Change Details**
Version 1.0.0	16 June 2016	First released version
	17 July 2016	Added swap16 and swap32 macros to handle byte ordering in 16 and 32 bit values

Contents

1. Overview 3

2. Files Supplied 3

3. Usage and Installation 3

3.1. Using libnewi2c.a static library 4

3.2. Using and Installing libnewi2c.so dynamic library 4

4. Using Makefile 4

4.1. Modify Makefile 4

4.2. Makefile targets 4

5. Description of the libnewi2c Library 6

5.1. Device and Register Addresses 6

5.2. I2CTypes 6

5.2.1. enum I2C_Result 6

5.2.2. #define I2C_BUFFER_SIZE 32 7

5.2.3. typedef struct I2C_Data 7

5.2.4. #define swap16(v) 7

5.2.5. #define swap32(v) 7

5.3. Class I2CAccess 7

5.3.1. I2CAccess Public Methods 7

5.3.1.1. static I2C_Result probe(unsigned char devAddr); 7

5.3.1.2. static I2C_Result write8(unsigned char devAddr, unsigned char val); 8

5.3.1.3. static I2C_Result write16(unsigned char devAddr, unsigned int val); 8

5.3.1.4. static I2C_Result write32(unsigned char devAddr, unsigned long val); 8

5.3.1.5. static I2C_Result writeBuffer(unsigned char devAddr, I2C_Data data); 8

5.3.1.6. static I2C_Result write8(unsigned char devAddr, unsigned char regAddr, unsigned char val); 9

5.3.1.7. static I2C_Result write16(unsigned char devAddr, unsigned char regAddr, unsigned int val); 9

5.3.1.8. static I2C_Result write32(unsigned char devAddr, unsigned char regAddr, unsigned long val); 9

5.3.1.9. static I2C_Result writeBuffer(unsigned char devAddr, unsigned char regAddr, I2C_Data data); 10

5.3.1.10. static I2C_Result read8(unsigned char devAddr, unsigned char & val); 10
5.3.1.11. static I2C_Result read16(unsigned char devAddr, unsigned int & val); 10
5.3.1.12. static I2C_Result read32(unsigned char devAddr, unsigned long & val); 11
5.3.1.13. static I2C_Result readBuffer(unsigned char devAddr, I2C_Data & data, int numBytes); 11
5.3.1.14. static I2C_Result read8(unsigned char devAddr, unsigned char regAddr, unsigned char & val); 11
5.3.1.15. static I2C_Result read16(unsigned char devAddr, unsigned char regAddr, unsigned int & val); 12
5.3.1.16. static I2C_Result read32(unsigned char devAddr, unsigned char regAddr, unsigned long & val); 12
5.3.1.17. static I2C_Result readBuffer(unsigned char devAddr, unsigned char regAddr, I2C_Data & data, int numBytes); 12
5.3.1.18. static void setRetryDelay(unsigned char devAddr, unsigned int delayMS); 13
5.3.1.19. static void setRetryCount(unsigned char devAddr, int count); 13
5.3.1.20. static unsigned int getRetryDelay(unsigned char devAddr); 13
5.3.1.21. static int getRetryCount(unsigned char devAddr); 13
5.3.1.22. static bool isAccessOk(); 14

5.4. Class I2CDevice 14

5.4.1. I2CDevice Constructor 14
5.4.1.1. Constructor - I2CDevice(unsigned char devAddr); 14
5.4.2. I2CDevice Public Methods 14
5.4.2.1. static I2C_Result probe(unsigned char devAddr); 14
5.4.2.2. I2C_Result write8(unsigned char val); 15
5.4.2.3. I2C_Result write16(unsigned int val); 15
5.4.2.4. I2C_Result write32(unsigned long val); 15
5.4.2.5. I2C_Result writeBuffer(I2C_Data data); 15
5.4.2.6. I2C_Result write8(unsigned char regAddr, unsigned char val); 16
5.4.2.7. I2C_Result write16(unsigned char regAddr, unsigned int val); 16
5.4.2.8. I2C_Result write32(unsigned char regAddr, unsigned long val); 16
5.4.2.9. I2C_Result writeBuffer(unsigned char regAddr, I2C_Data data); 16
5.4.2.10. I2C_Result read8(unsigned char & val); 17
5.4.2.11. I2C_Result read16(unsigned int & val); 17
5.4.2.12. I2C_Result read32(unsigned long & val); 17
5.4.2.13. I2C_Result readBuffer(I2C_Data & data, int numBytes); 18
5.4.2.14. I2C_Result read8(unsigned char regAddr, unsigned char & val); 18
5.4.2.15. I2C_Result read16(unsigned char regAddr, unsigned int & val); 18
5.4.2.16. I2C_Result read32(unsigned char regAddr, unsigned long & val); 18
5.4.2.17. I2C_Result readBuffer(unsigned char regAddr, I2C_Data & data, int numBytes); 19
5.4.2.18. void setRetryDelay(unsigned int delayMS); 19
5.4.2.19. void setRetryCount(int count); 19
5.4.2.20. unsigned int getRetryDelay(); 20
5.4.2.21. int getRetryCount(); 20
5.4.2.22. unsigned char getDevAddr(); 20

6. Further Development 20

1. Overview

libnewi2c is alternative C++ code for accessing the I2C devices connected to the Omega.

The rationale for producing this code to have proper C++ access to I2C devices as a replacement for the Onion provided code: https://github.com/OnionIoT/i2c-exp-driver/blob/master/src/lib/onion-i2c.c

libnewi2c consists of static and dynamic link libraries containing the classes used to interact with I2C devices.

These library and the main C++ classes are described in more details in this document.

The library was developed on a KUbuntu-14.04 system running in a VirtualBox VM and uses the OpenWrt toolchain for building the code:

The toolchain used can be found at:

- https://s3-us-west-2.amazonaws.com/onion-cdn/community/openwrt/OpenWrt-Toolchain-ar71xx-generic_gcc-4.8-linaro_uClibc-0.9.33.2.Linux-x86_64.tar.bz2

and details of its setup and usage can be found at:

- https://community.onion.io/topic/9/how-to-install-gcc/22

libnewi2c comes with **NO GUARANTEES** ☺ but you are free to use it and do what you want with it.

NOTE: Some of the code in the class **I2CAccess** as described below was derived from code to be found in the **onion-i2c.c** code for the Omega.

2. Files Supplied

libnewi2c is supplied in files in a GitHub repository at https://github.com/KitBishop/Omega-GPIO-I2C-Arduino/tree/master/libnewi2c. This repository contains the following important directories and files:

- **libnewi2c.pdf** – this documentation as a PDF file
- **Makefile** – the Makefile for **libnewi2c** library
- **hdr** – directory containing header (***.h**) files for **libnewi2c** library
- **src** – directory containing source (***.cpp**) files for **libnewi2c** library
- **bin** – directory containing the built library code:
    - **dynamic/libnewi2c.so** – the dynamic link version of the library
    - **static/libnewi2c.a** – the static link version of the library

3. Usage and Installation

Installing and using the library is simple. It primarily consists of linking your program that uses the library and for the dynamic link library, copying the library to a suitable location on your Omega.

3.1. Using libnewi2c.a static library

To use **libnewi2c.a** static library you simply need to statically link your program to that library file.

Then your program can be copied to and run on the Omega.

3.2. Using and Installing libnewi2c.so dynamic library

To use **libnewi2c.so** dynamic library you need to dynamically link your program to that library file.

For any program that uses **libnewi2c.so** the library file needs to be copied to the **/lib** directory on your Omega.

Alternatively, you can copy the library to any location that may be set up in any **LD_LIBRARY_PATH** directory on your Omega. For example, I use the following for testing:

- Created directory **/root/lib**
- Copied the library to **/root/lib**
- Added the following lines to my **/etc/profile** file:

```
LD_LIBRARY_PATH=/root/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

4. Using Makefile

A **Makefile** is supplied that can be used to build the library.

4.1. Modify Makefile

The **Makefile** will need modifying:

- You **NEED** to and **MUST** change **TOOL_BIN_DIR** to the "bin" directory of your OpenWrt uClibc toolchain. E.G. make appropriate change to **<xxxx>** in:

  TOOL_BIN_DIR=<xxxx>/OpenWrt-Toolchain-ar71xx-generic_gcc-4.8-linaro_uClibc-0.9.33.2.Linux-x86_64/toolchain-mips_34kc_gcc-4.8-linaro_uClibc-0.9.33.2/bin

4.2. Makefile targets

The **Makefile** implements the following set of targets:

- **make**
  The default target. Performs a complete build of both static and dynamic link versions of the library.
  This is directly equivalent to:
  make static dynamic

- **make static**
  Performs a complete build of just the static link version of the library.

- **make dynamic**
  Performs a complete build of just the dynamic link version of the library.

- **make clean**
  Removes all previous build files, both static and dynamic link versions.
  This is directly equivalent to:
  make clean-static clean-dynamic

- **make clean-static**
  Removes all previous build files for static link versions only
  .
- **make clean-dynamic**
  Removes all previous build files for dynamic link versions only

5. Description of the libnewi2c Library

The **libnewi2c** library contains three main components for access and usage of **libnewi2c** and some components that have internal usage only. These main components and their source files are:

- **I2CTypes** – defines a few basic types used elsewhere
  File:
  I2CTypes.h
- **I2CAccess** – a class used for direct access to the I2C hardware.
  Contains only **static** methods for access.
  Files:
  I2CAccess.h
  I2CAccess.cpp
- **I2CDevice** – a class used to represent instances of a single I2C device.
  Contains methods to interact with the specific device.
  Files:
  I2CDevice.h
  I2CDevice.cpp

The contents of these main components are described in following sections.

5.1. Device and Register Addresses

Communications with an I2C device always require a **device address**.

Some communications also require a **register address** on the device – this, and its usage, will be device specific and the documentation on the device itself will need to be consulted.

In the documentation below these are referenced as follows:

- **unsigned char devAddr** – the I2C address of the device. Due to the nature of I2C protocols, this will be a 7 bit address and must be in the range **0x03** to **0c7f**
- **unsigned char regAddr** – the register address on the device. This will be device specific. It is an 8 bit address in the range **0x00** to **0xff**

5.2. I2CTypes

The file **I2CTypes.h** contains definitions of some basic types used elsewhere.

5.2.1. enum I2C_Result

enum I2C_Result is used to represent the returned result of I2C operations. It has values:

- **I2C_OK = 0** - represents a successful result
- **I2C_BAD_ACCESS = 1** – indicates a failure accessing the I2C system in general
- **I2C_BAD_DEV = 2** – indicates a general failure accessing a specific I2C device
- **I2C_BAD_WRITE = 3** – indicates a failure writing to an I2C device
- **I2C_BAD_READ = 4** – indicates a failure reading from an I2C device

- **I2C_BAD_READ_ADDR = 5** – indicates a failure when sending an address to an I2C device prior to reading from it
- **I2C_BAD_READ_LEN = 6** – indicates that the amount of data read from an I2C device was less than that expected
- **I2C_BAD_PROBE = 7** – indicates a failure when performing a quick probe to test existence of an I2C device
- **I2C_TIME_OUT = 8** – indicates a timeout occurred while attempting to communicate to an I2C device

5.2.2. #define I2C_BUFFER_SIZE 32

Specifies the maximum size of data (32 bytes) that can be transferred to/from and I2C device in one operation.

5.2.3. typedef struct I2C_Data

Represents a structure used in transferring buffer data to/from and I2C device.

```
typedef struct I2C_Data {
    int size;
    unsigned char data[I2C_BUFFER_SIZE];
} I2C_Data;
```

NOTE:

- **size** – this is the number of bytes in the buffer data
- **data** – this is that actual buffer data

5.2.4. #define swap16(v)

Returns the 16 bit value of the parameter **v** with byte order reversed.

5.2.5. #define swap32(v)

Returns the 32 bit value of the parameter **v** with byte order reversed.

5.3. Class I2CAccess

The **I2CAccess** class is the main method by which all access is made to the I2C devices.

The class contains only static methods and no instance of this class will ever actually be created hence there are no constructors or destructors.

5.3.1. I2CAccess Public Methods

Note that in general, the success or failure of any method is returned as an **I2C_Result** value from each method.

5.3.1.1. static I2C_Result probe(unsigned char devAddr);

Performs a quick probe to a given I2C address to test for existence.

Parameters:

- **unsigned char devAddr** – the I2C address of the device

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.2. static I2C_Result write8(unsigned char devAddr, unsigned char val);

Writes an 8 bit value to an I2C address.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **unsigned char val** – the value to be written

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.3. static I2C_Result write16(unsigned char devAddr, unsigned int val);

Writes a 16 bit value to an I2C address.

NOTE: By default, 16 bit values are written least significant byte first. If it is required to send most significant byte first apply the macro **swap16** to the value to be written before writing.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **unsigned int val** – the value to be written

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.4. static I2C_Result write32(unsigned char devAddr, unsigned long val);

Writes a 32 bit value to an I2C address.

NOTE: By default, 32 bit values are written least significant byte first. If it is required to send most significant byte first apply the macro **swap32** to the value to be written before writing.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **unsigned long val** – the value to be written

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.5. static I2C_Result writeBuffer(unsigned char devAddr, I2C_Data data);

Write buffer data to an I2C address.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **I2C_Data data** – contains the data to be written and its length

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.6. static I2C_Result write8(unsigned char devAddr, unsigned char regAddr, unsigned char val);

Writes an 8 bit value to a given register on an I2C device.

Works by sending the register address followed by the value.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **unsigned char regAddr** – the register address on the device
- **unsigned char val** – the value to be written

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.7. static I2C_Result write16(unsigned char devAddr, unsigned char regAddr, unsigned int val);

Writes a 16 bit value to a given register on an I2C device.

Works by sending the register address followed by the value.

NOTE: By default, 16 bit values are written least significant byte first. If it is required to send most significant byte first apply the macro **swap16** to the value to be written before writing.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **unsigned char regAddr** – the register address on the device
- **unsigned int val** – the value to be written

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.8. static I2C_Result write32(unsigned char devAddr, unsigned char regAddr, unsigned long val);

Writes a 32 bit value to a given register on an I2C device.

Works by sending the register address followed by the value.

NOTE: By default, 32 bit values are written least significant byte first. If it is required to send most significant byte first apply the macro **swap32** to the value to be written before writing.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **unsigned char regAddr** – the register address on the device
- **unsigned long val** – the value to be written

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.9. static I2C_Result writeBuffer(unsigned char devAddr, unsigned char regAddr, I2C_Data data);

Writes buffer data to a given register on an I2C device.

Works by sending the register address followed by the buffer data.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **unsigned char regAddr** – the register address on the device
- **I2C_Data data** – contains the data to be written and its length

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.10. static I2C_Result read8(unsigned char devAddr, unsigned char & val);

Attempts to read an 8 bit value from an I2C address.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **unsigned char &val** – returns the value read

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.11. static I2C_Result read16(unsigned char devAddr, unsigned int & val);

Attempts to read a 16 bit value from an I2C address.

NOTE: By default, 16 bit values are read least significant byte first. If the value read has most significant byte first apply the macro **swap16** to the value read after reading.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **unsigned int &val** – returns the value read

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.12. static I2C_Result read32(unsigned char devAddr, unsigned long & val);

Attempts to read a 32 bit value from an I2C address.

NOTE: By default, 32 bit values are read least significant byte first. If the value read has most significant byte first apply the macro **swap32** to the value read after reading.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **unsigned long &val** – returns the value read

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.13. static I2C_Result readBuffer(unsigned char devAddr, I2C_Data & data, int numBytes);

Attempts to read a buffer of data from an I2C address.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **I2C_Data &data** – returns the buffer data read including length
- **int numBytes** – the number of bytes expected to be read

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.14. static I2C_Result read8(unsigned char devAddr, unsigned char regAddr, unsigned char & val);

Attempts to read an 8 bit value from a given register on an I2C address.

Works by first sending the register address to the device then attempting to read the data.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **unsigned char regAddr** – the register address on the device.
- **unsigned char &val** – returns the value read

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.15. static I2C_Result read16(unsigned char devAddr, unsigned char regAddr, unsigned int & val);

Attempts to read a 16 bit value from a given register on an I2C address.

Works by first sending the register address to the device then attempting to read the data.

NOTE: By default, 16 bit values are read least significant byte first. If the value read has most significant byte first apply the macro **swap16** to the value read after reading.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **unsigned char regAddr** – the register address on the device.
- **unsigned int &val** – returns the value read

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.16. static I2C_Result read32(unsigned char devAddr, unsigned char regAddr, unsigned long & val);

Attempts to read a 32 bit value from a given register on an I2C address.

Works by first sending the register address to the device then attempting to read the data.

NOTE: By default, 32 bit values are read least significant byte first. If the value read has most significant byte first apply the macro **swap32** to the value read after reading.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **unsigned char regAddr** – the register address on the device.
- **unsigned long &val** – returns the value read

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.17. static I2C_Result readBuffer(unsigned char devAddr, unsigned char regAddr, I2C_Data & data, int numBytes);

Attempts to read a 1buffer of data from a given register on an I2C address.

Works by first sending the register address to the device then attempting to read the data.

Parameters:

- **unsigned char devAddr** – the I2C address of the device

- **unsigned char regAddr** – the register address on the device.
- **I2C_Data &data** – returns the data read and its length
- **int numBytes** – the expected number of bytes to be read

Returns:

- Result of the call as an **I2C_Result** value

5.3.1.18. static void setRetryDelay(unsigned char devAddr, unsigned int delayMS);

Sets the retry delay on the I2C device. When accessing the device, the specified time will be waited before making any retries. The default delay is 1 millisecond.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **unsigned int delayMS** – the delay time in milliseconds between retries

Returns:

- <none>

5.3.1.19. static void setRetryCount(unsigned char devAddr, int count);

Sets the retry count on the I2C device. When accessing the device, any failed operation will be retried this number of times before the access is abandoned with a time out. The default retry count is 10.

Parameters:

- **unsigned char devAddr** – the I2C address of the device
- **int count** – the number of times to retry a failed access. If this value is less than 0, an unlimited number of retries will be performed.

Returns:

- <none>

5.3.1.20. static unsigned int getRetryDelay(unsigned char devAddr);

Returns the current retry delay on an I2C device.

Parameters:

- **unsigned char devAddr** – the I2C address of the device

Returns:

- the current retry delay in milliseconds

5.3.1.21. static int getRetryCount(unsigned char devAddr);

Returns the current retry count on an I2C device.

Parameters:

- **unsigned char devAddr** – the I2C address of the device

Returns:

- the current retry count

5.3.1.22. static bool isAccessOk();

Returns an indication as to whether or not general I2C access functionality is OK.

Returns the current retry count on an I2C device.

Parameters:

- <none>

Returns:

- **true** for access OK; false for access not OK

5.4. Class I2CDevice

The **I2CDevice** class represents instances of a single I2C device.

5.4.1. I2CDevice Constructor

5.4.1.1. Constructor - I2CDevice(unsigned char devAddr);

Creates a new I2CDevice instance for a given address.

Parameters:

- **unsigned char devAddr** – the device address for theI2Cdevice

5.4.2. I2CDevice Public Methods

Note that in general, the success or failure of any method is returned as an **I2C_Result** value from each method.

5.4.2.1. static I2C_Result probe(unsigned char devAddr);

Performs a quick probe to a given I2C address to test for existence.

Parameters:

- **unsigned char devAddr** – the I2C address of the device

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.2. I2C_Result write8(unsigned char val);

Writes an 8 bit value to the device.

Parameters:

- **unsigned char val** – the value to be written

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.3. I2C_Result write16(unsigned int val);

Writes a 16 bit value to the device.

NOTE: By default, 16 bit values are written least significant byte first. If it is required to send most significant byte first apply the macro **swap16** to the value to be written before writing.

Parameters:

- **unsigned int val** – the value to be written

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.4. I2C_Result write32(unsigned long val);

Writes a 32 bit value to the device.

NOTE: By default, 32 bit values are written least significant byte first. If it is required to send most significant byte first apply the macro **swap32** to the value to be written before writing.

Parameters:

- **unsigned long val** – the value to be written

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.5. I2C_Result writeBuffer(I2C_Data data);

Write buffer data to the device.

Parameters:

- **I2C_Data data** – contains the data to be written and its length

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.6. I2C_Result write8(unsigned char regAddr, unsigned char val);

Writes an 8 bit value to a given register on the device.

Works by sending the register address followed by the value.

Parameters:

- **unsigned char regAddr** – the register address on the device
- **unsigned char val** – the value to be written

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.7. I2C_Result write16(unsigned char regAddr, unsigned int val);

Writes a 16 bit value to a given register on the device.

Works by sending the register address followed by the value.

NOTE: By default, 16 bit values are written least significant byte first. If it is required to send most significant byte first apply the macro **swap16** to the value to be written before writing.

Parameters:

- **unsigned char regAddr** – the register address on the device
- **unsigned int val** – the value to be written

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.8. I2C_Result write32(unsigned char regAddr, unsigned long val);

Writes a 32 bit value to a given register on the device.

Works by sending the register address followed by the value.

NOTE: By default, 32 bit values are written least significant byte first. If it is required to send most significant byte first apply the macro **swap32** to the value to be written before writing.

Parameters:

- **unsigned char regAddr** – the register address on the device
- **unsigned long val** – the value to be written

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.9. I2C_Result writeBuffer(unsigned char regAddr, I2C_Data data);

Writes buffer data to a given register on the device.

Works by sending the register address followed by the buffer data.

Parameters:

- **unsigned char regAddr** – the register address on the device
- **I2C_Data data** – contains the data to be written and its length

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.10. I2C_Result read8(unsigned char & val);

Attempts to read an 8 bit value from the device.

Parameters:

- **unsigned char &val** – returns the value read

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.11. I2C_Result read16(unsigned int & val);

Attempts to read a 16 bit value from the device.

NOTE: By default, 16 bit values are read least significant byte first. If the value read has most significant byte first apply the macro **swap16** to the value read after reading.

Parameters:

- **unsigned int &val** – returns the value read

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.12. I2C_Result read32(unsigned long & val);

Attempts to read a 32 bit value from the device.

NOTE: By default, 32 bit values are read least significant byte first. If the value read has most significant byte first apply the macro **swap32** to the value read after reading.

Parameters:

- **unsigned long &val** – returns the value read

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.13. I2C_Result readBuffer(I2C_Data & data, int numBytes);

Attempts to read a buffer of data from the device.

Parameters:

- **I2C_Data &data** – returns the buffer data read including length
- **int numBytes** – the number of bytes expected to be read

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.14. I2C_Result read8(unsigned char regAddr, unsigned char & val);

Attempts to read an 8 bit value from a given register on the device.

Works by first sending the register address to the device then attempting to read the data.

Parameters:

- **unsigned char regAddr** – the register address on the device.
- **unsigned char &val** – returns the value read

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.15. I2C_Result read16(unsigned char regAddr, unsigned int & val);

Attempts to read a 16 bit value from a given register on the device.

Works by first sending the register address to the device then attempting to read the data.

NOTE: By default, 16 bit values are read least significant byte first. If the value read has most significant byte first apply the macro **swap16** to the value read after reading.

Parameters:

- **unsigned char regAddr** – the register address on the device.
- **unsigned int &val** – returns the value read

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.16. I2C_Result read32(unsigned char regAddr, unsigned long & val);

Attempts to read a 32 bit value from a given register on the device.

Works by first sending the register address to the device then attempting to read the data.

NOTE: By default, 32 bit values are read least significant byte first. If the value read has most significant byte first apply the macro **swap32** to the value read after reading.

Parameters:

- **unsigned char regAddr** – the register address on the device.
- **unsigned long &val** – returns the value read

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.17. I2C_Result readBuffer(unsigned char regAddr, I2C_Data & data, int numBytes);

Attempts to read a 1buffer of data from a given register on the device.

Works by first sending the register address to the device then attempting to read the data.

Parameters:

- **unsigned char regAddr** – the register address on the device.
- **I2C_Data &data** – returns the data read and its length
- **int numBytes** – the expected number of bytes to be read

Returns:

- Result of the call as an **I2C_Result** value

5.4.2.18. void setRetryDelay(unsigned int delayMS);

Sets the retry delay on the device. When accessing the device, the specified time will be waited before making any retries. The default delay is 1 millisecond.

Parameters:

- **unsigned int delayMS** – the delay time in milliseconds between retries

Returns:

- <none>

5.4.2.19. void setRetryCount(int count);

Sets the retry count on the device. When accessing the device, any failed operation will be retried this number of times before the access is abandoned with a time out. The default retry count is 10.

Parameters:

- **int count** – the number of times to retry a failed access. If this value is less than 0, an unlimited number of retries will be performed.

Returns:

- <none>

5.4.2.20. unsigned int getRetryDelay();

Returns the current retry delay on the device.

Parameters:

- <none>

Returns:

- the current retry delay in milliseconds

5.4.2.21. int getRetryCount();

Returns the current retry count on the device.

Parameters:

- <none>

Returns:

- the current retry count

5.4.2.22. unsigned char getDevAddr();

Returns the I2C device address of the device.

Parameters:

- <none>

Returns:

- the I2C device address

6. Further Development

Development of **libnewi2c** is on-going. There will be changes and additions to the code in the future.