MLX90614 Device Driver

1.0

Generated by Doxygen 1.8.14

Tue Sep 17 2019 11:39:01

Contents

1 Arduino Library for the MLX90614 Temperature Sensor

This library was written to enable remote sensing of the temperature of the rotors of outrunner style brushless DC motors used in remotely piloted aircraft, for the purpose of real time data logging and air to ground telemetry.

These sensors use the SMB bus protocol to communicate. This is similar, though not identical, to the I2C bus. There is enough similarity to enable the Arduino standard Wire library to communicate with the device, however not all features can be implemented, for example it is not possible to read the flags register with standard Wire functions. 2 pins are required to interface the device to an Arduino - the SDA and SCL lines.

Installing

Download the distribution package and decompress it.
Rename the uncompressed folder ***/mlx90614***.
Check that the ***/mlx90614*** folder contains the following files;

MLX90614.cpp
MLX90614.h
MLX90614.chm
MLX90614.pdf
Crc8.cpp
Crc8.h
property.h
doxyfile

Place the ***/mlx90614*** library folder into your ***arduinosketchfolder/libraries/*** folder.
You may need to create the libraries subfolder if its your first library. Restart the IDE.

Documentation

MLX90614.chm and *MLX90614.pdf* contain the documentation for the classes.
A Doxygen script is included to enable generation of documentation. You will need the graph tool, the dot tool, and the help compiler, in addition to editing the paths to these tools in the script to suit your environment.

Author

John Fitter B.E., Eagle Air Australia Pty. Ltd.
This library was inspired by a library written by Adafruit Industries.

License

2 GNU GENERAL PUBLIC LICENSE

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either
  - (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or
  - (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

3 Class Index

3.1 Class List

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

4 File Index

4.1 File List

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

5 Class Documentation

5.1 CRC8 Class Reference

Public Member Functions

- CRC8 (uint8_t polynomial=CRC8_DEFAULTPOLY)

  CRC8 class constructor.
- uint8_t crc8 (void)

  Return the current value of the CRC.
- uint8_t crc8 (uint8_t data)

  Update the current value of the CRC.
- void crc8Start (uint8_t poly)

  Initialize the CRC8 object.

Private Attributes

- uint8_t **_crc**
- uint8_t **_poly**

5.1.1 Detailed Description

Definition at line 37 of file Crc8.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 CRC8()

```
CRC8::CRC8 (
                uint8_t poly = CRC8_DEFAULTPOLY )
```

CRC8 class constructor.

Parameters

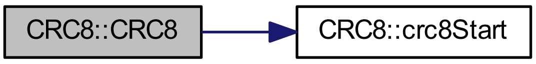
in	*poly*	8 bit CRC polynomial to use.

Definition at line 36 of file Crc8.cpp.

References crc8Start().

```
00036 {crc8Start(poly);}
```

Here is the call graph for this function:



5.1.3 Member Function Documentation

5.1.3.1 crc8() [1/2]

```
uint8_t CRC8::crc8 (
                void )
```

Return the current value of the CRC.

Returns

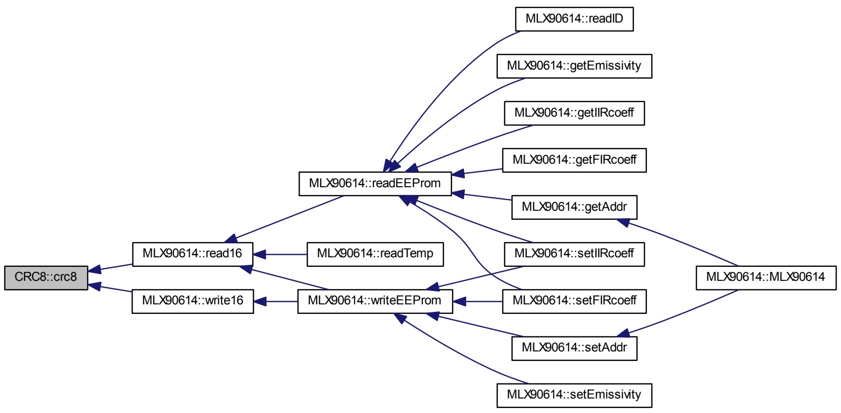
8 bit CRC current value.

Definition at line 42 of file Crc8.cpp.

Referenced by MLX90614::read16(), and MLX90614::write16().

```
00042 {return _crc;}
```

Here is the caller graph for this function:



5.1.3.2 crc8() [2/2]

```
uint8_t CRC8::crc8 (
            uint8_t data )
```

Update the current value of the CRC.

Parameters

in	*data*	New 8 bit data to be added to the CRC.

Returns

8 bit CRC current value.

Definition at line 49 of file Crc8.cpp.

```
00049                              {
00050     uint8_t i = 8;
00051
00052     _crc ^= data;
00053     while(i--) _crc = _crc & 0x80 ? (_crc << 1) ^ _poly : _crc << 1;
00054     return _crc;
00055 }
```

5.1.3.3 crc8Start()

```
void CRC8::crc8Start (
             uint8_t poly )
```

Initialize the CRC8 object.

Parameters

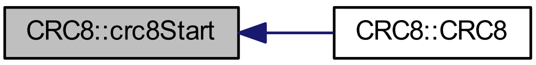
in	*poly*	8 bit CRC polynomial to use.

Definition at line 61 of file Crc8.cpp.

Referenced by CRC8().

```
00061                                  {
00062     _poly = poly;
00063     _crc = 0;
00064 }
```

Here is the caller graph for this function:



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

- Crc8.h
- Crc8.cpp

5.2 defaultEEPromData Struct Reference

EEPROM memory contents factory default values.

Public Attributes

- uint8_t **address**
- uint16_t **data**

5.2.1 Detailed Description

EEPROM memory contents factory default values.

Definition at line 165 of file MelexisTest.ino.

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

- MelexisTest.ino

5.3 MLX90614 Class Reference

Public Types

- enum tempUnit_t { MLX90614_TK, MLX90614_TC, MLX90614_TF }
- enum tempSrc_t { MLX90614_SRCA, MLX90614_SRC01, MLX90614_SRC02 }

Public Member Functions

- MLX90614 (uint8_t i2caddr=MLX90614_I2CDEFAULTADDR)

  MLX90614 Device class constructor.
- boolean begin ()

  Initialize the device and the i2c interface.
- boolean **isReady** (void)
- uint64_t readID (void)

  Retrieve the chip ID bytes.
- uint8_t getIIRcoeff (void)

  Get the coefficients of the IIR digital filter.
- uint8_t getFIRcoeff (void)

  Get the coefficients of the FIR digital filter.
- float getEmissivity (void)

  Get the emissivity of the object.
- void setIIRcoeff (uint8_t csb=4)

  Set the coefficients of the IIR digital filter.
- void setFIRcoeff (uint8_t csb=7)

  Set the coefficients of the FIR digital filter.
- void setEmissivity (float emiss=1.0)

  Set the emissivity of the object.
- uint16_t readEEProm (uint8_t)

  Return a 16 bit value read from EEPROM.
- void writeEEProm (uint8_t, uint16_t)

  Write a 16 bit value to EEPROM after first clearing the memory.
- double readTemp (tempSrc_t=MLX90614_SRC01, tempUnit_t=MLX90614_TC)

  Return a temperature from the specified source in specified units.
- double convKtoC (double)

  Convert temperature in degrees K to degrees C.
- double convCtoF (double)

  Convert temperature in degrees C to degrees F.

Public Attributes

- Property< uint8_t, MLX90614 > busAddr
- Property< uint8_t, MLX90614 > rwError
- Property< uint8_t, MLX90614 > crc8
- Property< uint8_t, MLX90614 > pec

Private Member Functions

- uint16_t read16 (uint8_t)

    Return a 16 bit value read from RAM or EEPROM.
- void write16 (uint8_t, uint16_t)

    Write a 16 bit value to memory.
- uint8_t getRwError (void)
- uint8_t getCRC8 (void)
- uint8_t getPEC (void)
- uint8_t getAddr (void)

    Return the device SMBus address.
- void setAddr (uint8_t)

    Set device SMBus address.

Private Attributes

- boolean **_ready**
- uint8_t _addr
- uint8_t _rwError
- uint8_t _crc8
- uint8_t _pec

5.3.1 Detailed Description

Definition at line 104 of file MLX90614.h.

5.3.2 Member Enumeration Documentation

5.3.2.1 tempSrc_t

```
enum MLX90614::tempSrc_t
```

Enumerations for temperature measurement source.

Enumerator

MLX90614_SRCA	Chip (ambient) sensor
MLX90614_SRC01	IR source #1
MLX90614_SRC02	IR source #2

Definition at line 134 of file MLX90614.h.

```
        :
    boolean  _ready;
```

5.3.2.2 tempUnit_t

```
enum MLX90614::tempUnit_t
```

Enumerations for temperature units.

Enumerator

MLX90614_TK	degrees Kelvin
MLX90614_TC	degrees Centigrade
MLX90614_TF	degrees Fahrenheit

Definition at line 129 of file MLX90614.h.

```
                         {MLX90614_SRCA,                        /**< Chip (ambient) sensor */
                          MLX90614_SRC01,                       /**< IR source #1 */
```

5.3.3 Constructor & Destructor Documentation

5.3.3.1 MLX90614()

```
MLX90614::MLX90614 (
            uint8_t i2caddr = MLX90614_I2CDEFAULTADDR )
```

MLX90614 Device class constructor.

Parameters

in	*i2caddr*	Device address (default: published value).

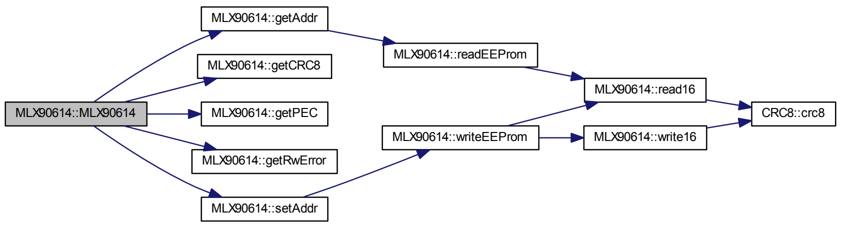
Definition at line 46 of file MLX90614.cpp.

References _addr, busAddr, crc8, getAddr(), getCRC8(), getPEC(), getRwError(), pec, rwError, and setAddr().

```
                                   {

    busAddr.Set_Class(this);
    busAddr.Set_Get(&MLX90614::getAddr);
    busAddr.Set_Set(&MLX90614::setAddr);

    rwError.Set_Class(this);
    rwError.Set_Get(&MLX90614::getRwError);

```

```
00055     pec.Set_Class(this);
00056     pec.Set_Get(&MLX90614::getPEC);
00057
00058     crc8.Set_Class(this);
00059     crc8.Set_Get(&MLX90614::getCRC8);
00060
00061     _addr = i2caddr;
00062     _ready = false;
00063 }
```

Here is the call graph for this function:



5.3.4 Member Function Documentation

5.3.4.1 convCtoF()

```
double MLX90614::convCtoF (
              double degC )
```

Convert temperature in degrees C to degrees F.

Parameters

in	*degC*	Temperature in degrees Centigrade.

Returns

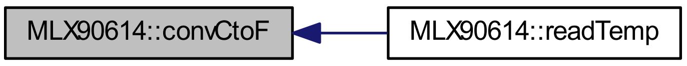
Temperature in degrees Fahrenheit.

Definition at line 389 of file MLX90614.cpp.

Referenced by readTemp().

```
00389 {return (degC * 1.8) + 32.0;}
```

Here is the caller graph for this function:



5.3.4.2 convKtoC()

```
double MLX90614::convKtoC (
             double degK )
```

Convert temperature in degrees K to degrees C.

Parameters

in	*degK*	Temperature in degrees Kelvin.

Returns

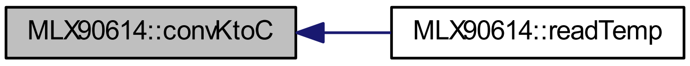
Temperature in degrees Centigrade.

Definition at line 382 of file MLX90614.cpp.

Referenced by readTemp().

```
00382 {return degK - 273.15;}
```

Here is the caller graph for this function:



5.3.4.3 getAddr()

```
uint8_t MLX90614::getAddr (
            void  )  [private]
```

Return the device SMBus address.

SMB bus address getter

Remarks

- Must be only device on the bus.
- Sets the library to use the new found address.

Returns

Device address.

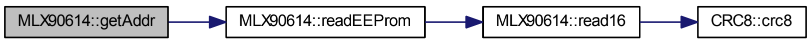
Definition at line 250 of file MLX90614.cpp.

References _addr, _rwError, and readEEProm().

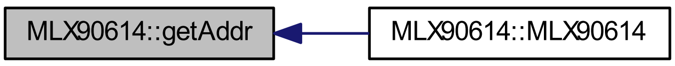
Referenced by MLX90614().

```
                                 {
    uint8_t tempAddr = _addr;

    _rwError = 0;

    // It is assumed we do not know the existing slave address so the broadcast address is used.
    // This will throw a r/w error so errors will be ignored.
    _addr = MLX90614_BROADCASTADDR;

    // Reload program copy with the existing slave address.
    _addr = lowByte(readEEProm(MLX90614_ADDR));

    return _addr;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.4.4 getCRC8()

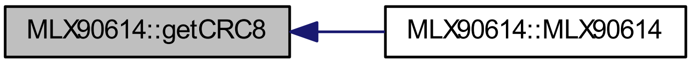
```
uint8_t MLX90614::getCRC8 (
            void ) [inline], [private]
```

8 bit CRC getter

Definition at line 154 of file MLX90614.h.

Referenced by MLX90614().

Here is the caller graph for this function:



5.3.4.5 getEmissivity()

```
float MLX90614::getEmissivity (
            void )
```

Get the emissivity of the object.

Emissivity getter

Remarks

The emissivity is stored as a 16 bit integer defined by the following:
`emissivity = dec2hex[round(65535 x emiss)]`

Returns

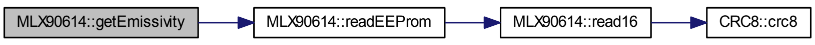
Physical emissivity value in range 0.1 ...1.0

Definition at line 120 of file MLX90614.cpp.

References _rwError, and readEEProm().

```
00120                               {
00121
00122     _rwError = 0;
00123     uint16_t emiss = readEEProm(MLX90614_EMISS);
00124     if(_rwError) return (float)1.0;
00125     return (float)emiss / 65535.0;
00126 }
```

Here is the call graph for this function:



5.3.4.6 getFIRcoeff()

```
uint8_t MLX90614::getFIRcoeff (
            void )
```

Get the coefficients of the FIR digital filter.

IIR coefficient getter

Remarks

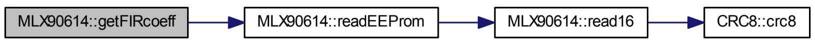
The FIR digital filter coefficient N is bits 10:8 of ConfigRegister1
The value of N is set as follows: `N = 2 ^ (csb + 3)`
The manufacturer does not recommend `N < 128`

Definition at line 208 of file MLX90614.cpp.

References _rwError, and readEEProm().

```
                              {

    _rwError = 0;

    // Get the current value of ConfigRegister1 bits 10:8
    uint8_t fir = (readEEProm(MLX90614_CONFIG) >> 8) & 7;

    if(_rwError) return 7;
    return fir;
}
```

Here is the call graph for this function:



5.3.4.7 getIIRcoeff()

```
uint8_t MLX90614::getIIRcoeff (
            void )
```

Get the coefficients of the IIR digital filter.

IIR coefficient getter

Remarks

The IIR digital filter coefficients are set by the LS 3 bits of ConfigRegister1

Returns

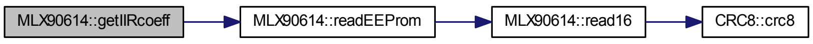
Filter coefficient table index. Range 0...7

Definition at line 166 of file MLX90614.cpp.

References _rwError, and readEEProm().

```
                                    {

    _rwError = 0;

    // Get the current value of ConfigRegister1 bits 2:0
    uint8_t iir = readEEProm(MLX90614_CONFIG) & 7;

    if(_rwError) return 4;
    return iir;
}
```

Here is the call graph for this function:



5.3.4.8 getPEC()

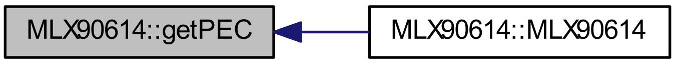
```
uint8_t MLX90614::getPEC (
              void  )  [inline], [private]
```

PEC getter

Definition at line 155 of file MLX90614.h.

Referenced by MLX90614().

Here is the caller graph for this function:



5.3.4.9 getRwError()

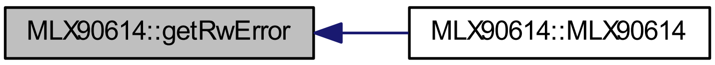
```
uint8_t MLX90614::getRwError (
            void  )  [inline], [private]
```

R/W error flags getter

Definition at line 153 of file MLX90614.h.

Referenced by MLX90614().

Here is the caller graph for this function:



5.3.4.10 read16()

```
uint16_t MLX90614::read16 (
            uint8_t cmd )  [private]
```

Return a 16 bit value read from RAM or EEPROM.

Parameters

in	*cmd*	Command to send (register to read from).

Returns

Value read from memory.

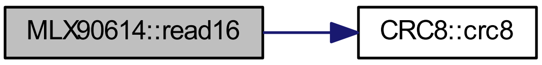
Definition at line 270 of file MLX90614.cpp.

References _addr, _crc8, _pec, _rwError, and CRC8::crc8().

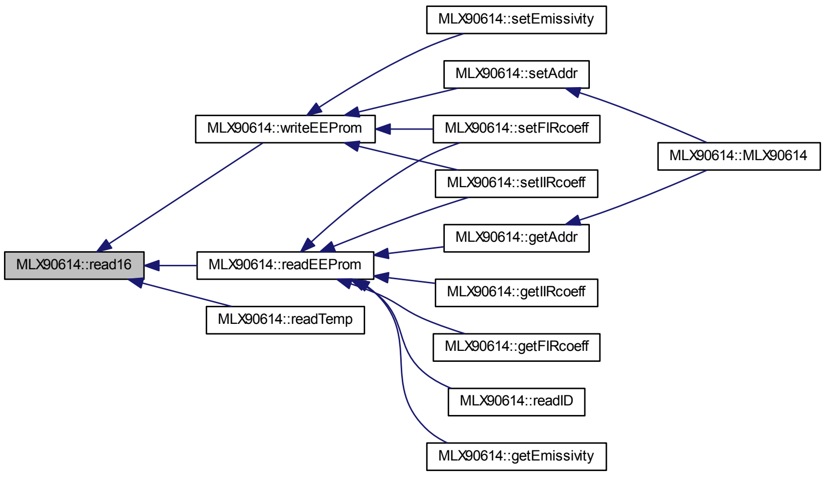
Referenced by readEEProm(), readTemp(), and writeEEProm().

```
                                     {
    uint16_t val;
    CRC8 crc(MLX90614_CRC8POLY);

    // Send the slave address then the command and set any error status bits returned by the write.
    Wire.beginTransmission(_addr);
    Wire.write(cmd);
    _rwError |= (1 << Wire.endTransmission(false)) >> 1;

    // Experimentally determined delay to prevent read errors (manufacturer's data sheet has
    // left something out).
    delayMicroseconds(MLX90614_XDLY);

    // Resend slave address then get the 3 returned bytes.
    Wire.requestFrom(_addr, (uint8_t)3);

    // Data is returned as 2 bytes little endian.
    val = Wire.read();
    val |= Wire.read() << 8;

    // Rread the PEC (CRC-8 of all bytes).
    _pec = Wire.read();

    // Clear r/w errors if using broadcast address.
    if(_addr == MLX90614_BROADCASTADDR) _rwError &= MLX90614_NORWERROR;

    // Build our own CRC-8 of all received bytes.
    crc.crc8(_addr << 1);
    crc.crc8(cmd);
    crc.crc8((_addr << 1) + 1);
    crc.crc8(lowByte(val));
    _crc8 = crc.crc8(highByte(val));

    // Set error status bit if CRC mismatch.
    if(_crc8 != _pec) _rwError |= MLX90614_RXCRC;

    return val;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.4.11 readEEProm()

```
uint16_t MLX90614::readEEProm (
            uint8_t addr )
```

Return a 16 bit value read from EEPROM.

Parameters

in	*addr*	Register address to read from.

Returns

Value read from EEPROM.

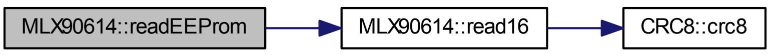
Definition at line 344 of file MLX90614.cpp.

References read16().

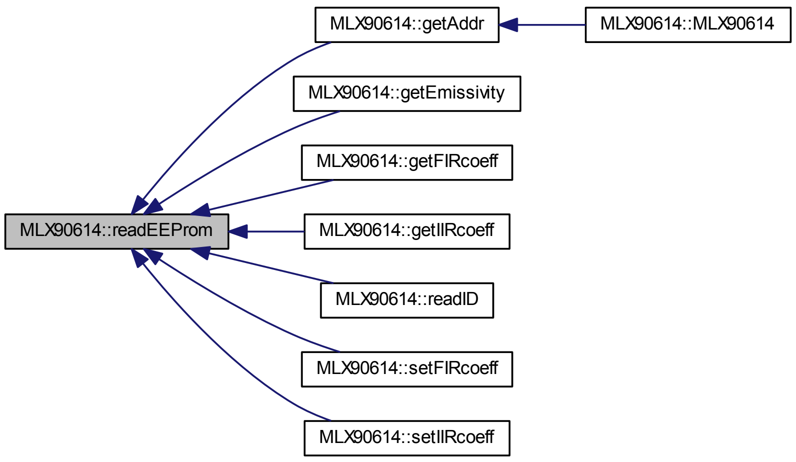
Referenced by getAddr(), getEmissivity(), getFIRcoeff(), getIIRcoeff(), readID(), setFIRcoeff(), and setIIRcoeff().

```
00344 {return read16(addr | 0x20);}
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.4.12 readID()

```
uint64_t MLX90614::readID (
            void   )
```

Retrieve the chip ID bytes.

Chip ID getter

Returns

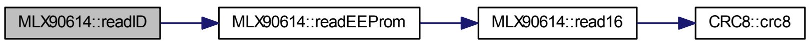
Chip ID as a 64 bit word.

Definition at line 395 of file MLX90614.cpp.

References readEEProm().

```
                              {
    uint64_t ID = 0;

    // If we are lucky the compiler will optimise this.
    for(uint8_t i = 0; i < 4; i++) ID = (ID <<= 16) | readEEProm(MLX90614_ID1 + i);
    return ID;
}
```

Here is the call graph for this function:



5.3.4.13 readTemp()

```
double MLX90614::readTemp (
            tempSrc_t tsrc = MLX90614_SRC01,
            tempUnit_t tunit = MLX90614_TC )
```

Return a temperature from the specified source in specified units.

Remarks

- Temperature is stored in ram as a 16 bit absolute value to a resolution of 0.02K
- Linearized sensor die temperature is available as Ta (ambient).
- One or two object temperatures are linearized to the range -38.2C...125C

Parameters

in	*tsrc*	Internal temperature source to read, default #1.
in	*tunit*	Temperature units to convert raw data to, default deg Celsius.

Returns

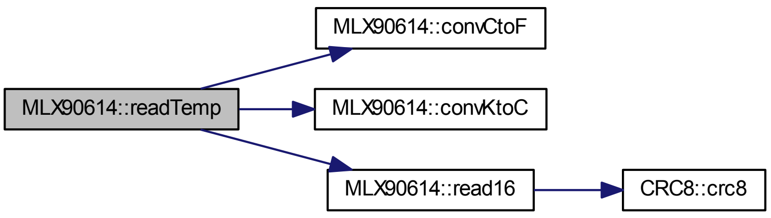
Temperature.

Definition at line 84 of file MLX90614.cpp.

References _rwError, convCtoF(), convKtoC(), MLX90614_SRC01, MLX90614_SRC02, MLX90614_TC, MLX90614_TF, and read16().

```
                                                     {
    double temp;

    _rwError = 0;
    switch(tsrc) {
        case MLX90614_SRC01 : temp = read16(MLX90614_TOBJ1); break;
        case MLX90614_SRC02 : temp = read16(MLX90614_TOBJ2); break;
        default : temp = read16(MLX90614_TA);
    }
    temp *= 0.02;
    switch(tunit) {
        case MLX90614_TC : return convKtoC(temp);
        case MLX90614_TF : return convKtoC(convCtoF(temp));
    }
    return temp;
}
```

Here is the call graph for this function:



5.3.4.14 setAddr()

```
void MLX90614::setAddr (
              uint8_t addr )  [private]
```

Set device SMBus address.

SMB bus address setter

Remarks

- Must be only device on the bus.
- Must power cycle the device after changing address.

Parameters

in	*addr*	New device address. Range 1...127

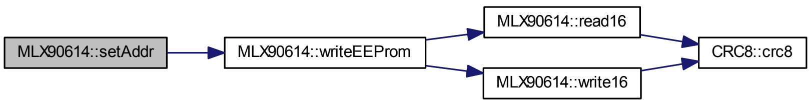
Definition at line 226 of file MLX90614.cpp.

References _addr, _rwError, and writeEEProm().

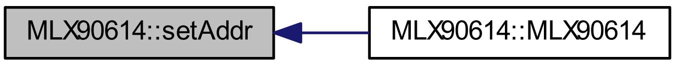
Referenced by MLX90614().

```
                                {

    _rwError = 0;

    // It is assumed we do not know the existing slave address so the broadcast address is used.
    // First ensure the new address is in the legal range (1..127)
    if(addr &= 0x7f) {
        _addr = MLX90614_BROADCASTADDR;
        writeEEProm(MLX90614_ADDR, addr);

        // There will always be a r/w error using the broadcast address so we cannot respond
        // to r/w errors. We must just assume this worked.
        _addr = addr;

    } else _rwError |= MLX90614_INVALIDATA;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.4.15 setEmissivity()

```
void MLX90614::setEmissivity (
                float emiss = 1.0 )
```

Set the emissivity of the object.

Emissivity setter

Remarks

The emissivity is stored as a 16 bit integer defined by the following:
`emissivity = dec2hex[round(65535 x emiss)]`

Parameters

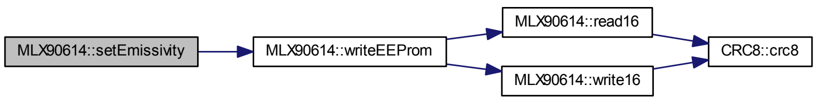
in	*emiss*	Physical emissivity value in range 0.1 ...1.0, default 1.0

Definition at line 107 of file MLX90614.cpp.

References _rwError, and writeEEProm().

```
                                        {
    _rwError = 0;
    uint16_t e = int(emiss * 65535. + 0.5);
    if((emiss > 1.0) || (e < 6553)) _rwError |= MLX90614_INVALIDATA;
    else writeEEProm(MLX90614_EMISS, e);
}
```

Here is the call graph for this function:



5.3.4.16 setFIRcoeff()

```
void MLX90614::setFIRcoeff (
             uint8_t csb = 7 )
```

Set the coefficients of the FIR digital filter.

IIR coefficient setter

Remarks

The FIR digital filter coefficient N is bits 10:8 of ConfigRegister1
The value of N is set as follows: `N = 2 ^ (csb + 3)`
The manufacturer does not recommend `N < 128`

Parameters

in	*csb*	See page 12 of datasheet. Range 0...7, default = 7 (N = 1024)

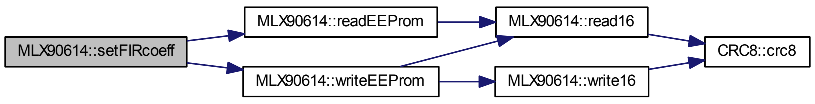
Definition at line 184 of file MLX90614.cpp.

References _rwError, readEEProm(), and writeEEProm().

```
                                      {
```

```

    _rwError = 0;

    // Ensure legal range by clearing all but the LS 3 bits.
    csb &= 7;

    // Get the current value of ConfigRegister1
    uint16_t reg = readEEProm(MLX90614_CONFIG);

    // Clear bits 10:8, mask in the new value, then write it back.
    if(!_rwError) {
        reg &= 0xf8ff;
        reg |= (uint16_t)csb << 8;
        writeEEProm(MLX90614_CONFIG, reg);
    }
}
```

Here is the call graph for this function:



5.3.4.17 setIIRcoeff()

```
void MLX90614::setIIRcoeff (
             uint8_t csb = 4 )
```

Set the coefficients of the IIR digital filter.

IIR coefficient setter

Remarks

The IIR digital filter coefficients are set by the LS 3 bits of ConfigRegister1
The value of the coefficients is set as follows:

```
csb = 0   a1 = 0.5    a2 = 0.5
      1        0.25        0.75
      2        0.167       0.833
      3        0.125       0.875
      4        1           0  (IIR bypassed)
      5        0.8         0.2
      6        0.67        0.33
      7        0.57        0.43
```

Parameters

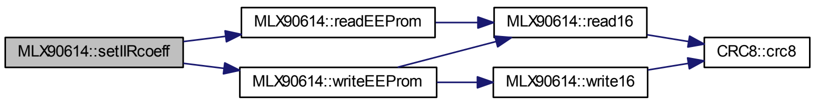
in	*csb*	See page 12 of datasheet. Range 0...7, default = 4 (IIR bypassed)

Definition at line 143 of file MLX90614.cpp.

References _rwError, readEEProm(), and writeEEProm().

```
                                          {

    _rwError = 0;

    // Ensure legal range by clearing all but the LS 3 bits.
    csb &= 7;

    // Get the current value of ConfigRegister1
    uint16_t reg = readEEProm(MLX90614_CONFIG);

    // Clear bits 2:0, mask in the new value, then write it back.
    if(!_rwError) {
        reg &= 0xfff8;
        reg |= (uint16_t)csb;
        writeEEProm(MLX90614_CONFIG, reg);
    }
}
```

Here is the call graph for this function:



5.3.4.18 write16()

```
void MLX90614::write16 (
              uint8_t cmd,
              uint16_t data )  [private]
```

Write a 16 bit value to memory.

Parameters

in	*cmd*	Command to send (register to write to).
in	*data*	Value to write.

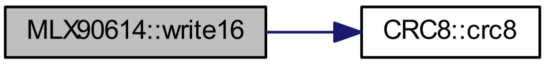
Definition at line 314 of file MLX90614.cpp.

References _addr, _crc8, _pec, _rwError, and CRC8::crc8().

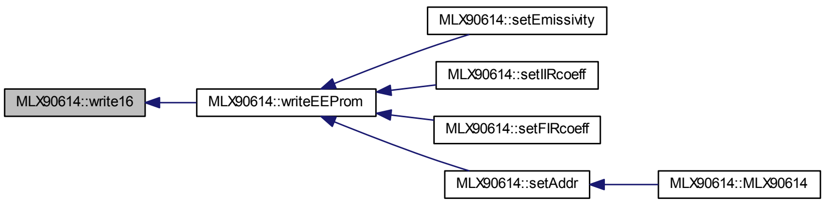
Referenced by writeEEProm().

```
                                                         {
    CRC8 crc(MLX90614_CRC8POLY);

    // Build the CRC-8 of all bytes to be sent.
    crc.crc8(_addr << 1);
    crc.crc8(cmd);
    crc.crc8(lowByte(data));
    _crc8 = crc.crc8(highByte(data));

    // Send the slave address then the command.
    Wire.beginTransmission(_addr);
    Wire.write(cmd);

    // Write the data low byte first.
    Wire.write(lowByte(data));
    Wire.write(highByte(data));

    // Then write the crc and set the r/w error status bits.
    Wire.write(_pec = _crc8);
    _rwError |= (1 << Wire.endTransmission(true)) >> 1;

    // Clear r/w errors if using broadcast address.
    if(_addr == MLX90614_BROADCASTADDR) _rwError &= MLX90614_NORWERROR;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.4.19 writeEEProm()

```
void MLX90614::writeEEProm (
            uint8_t reg,
            uint16_t data )
```

Write a 16 bit value to EEPROM after first clearing the memory.

Remarks

- Erase and write time 5ms per manufacturer specification
- Manufacturer does not specify max or min erase/write times

Parameters

in	*reg*	Address to write to.
in	*data*	Value to write.

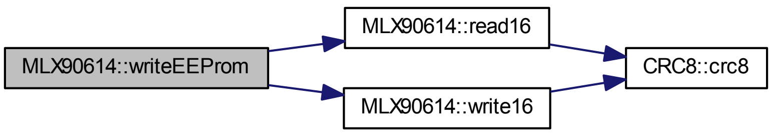
Definition at line 354 of file MLX90614.cpp.

References _rwError, read16(), and write16().

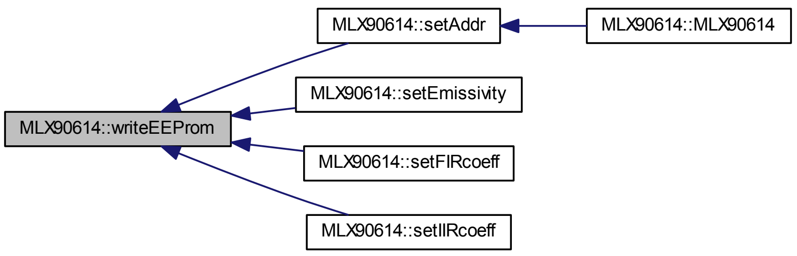
Referenced by setAddr(), setEmissivity(), setFIRcoeff(), and setIIRcoeff().

```
00354                                                       {
00355     uint16_t val;
00356     reg |= 0x20;
00357 
00358     // Read current value, compare to the new value, and do nothing on a match or if there are
00359     // read errors set the error status flag only.
00360     val = read16(reg);
00361     if((val != data) && !_rwError) {
00362 
00363         // On any R/W errors it is assumed the memory is corrupted.
00364         // Clear the memory and wait Terase (per manufacturer's documentation).
00365         write16(reg, 0);
00366         delay(5);
00367         if(_rwError) _rwError |= MLX90614_EECORRUPT;
00368 
00369         // Write the data and wait Twrite (per manufacturer's documentation)
00370         // and set the r/w error status bits.
00371         write16(reg, data);
00372         delay(5);
00373         if(_rwError) _rwError |= MLX90614_EECORRUPT;
00374     }
00375 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.5 Member Data Documentation

5.3.5.1 _addr

```
uint8_t MLX90614::_addr  [private]
```

Slave address

Definition at line 145 of file MLX90614.h.

Referenced by getAddr(), MLX90614(), read16(), setAddr(), and write16().

5.3.5.2 _crc8

```
uint8_t MLX90614::_crc8  [private]
```

8 bit CRC

Definition at line 147 of file MLX90614.h.

Referenced by begin(), read16(), and write16().

5.3.5.3 _pec

```
uint8_t MLX90614::_pec  [private]
```

PEC

Definition at line 148 of file MLX90614.h.

Referenced by begin(), read16(), and write16().

5.3.5.4 _rwError

```
uint8_t MLX90614::_rwError  [private]
```

R/W error flags

Definition at line 146 of file MLX90614.h.

Referenced by begin(), getAddr(), getEmissivity(), getFIRcoeff(), getIIRcoeff(), read16(), readTemp(), setAddr(), setEmissivity(), setFIRcoeff(), setIIRcoeff(), write16(), and writeEEProm().

5.3.5.5 busAddr

```
Property<uint8_t, MLX90614> MLX90614::busAddr
```

SMBus address property

Definition at line 123 of file MLX90614.h.

Referenced by MLX90614().

5.3.5.6 crc8

`Property<uint8_t, MLX90614> MLX90614::crc8`

8 bit CRC property

Definition at line 125 of file MLX90614.h.

Referenced by MLX90614().

5.3.5.7 pec

`Property<uint8_t, MLX90614> MLX90614::pec`

PEC property

Definition at line 126 of file MLX90614.h.

Referenced by MLX90614().

5.3.5.8 rwError

`Property<uint8_t, MLX90614> MLX90614::rwError`

R/W error flags property

Definition at line 124 of file MLX90614.h.

Referenced by MLX90614().

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

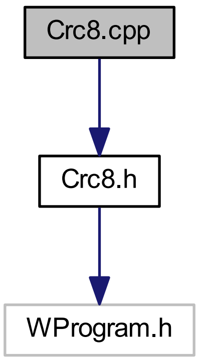
- MLX90614.h
- MLX90614.cpp

6 File Documentation

6.1 Crc8.cpp File Reference

8 bit CRC helper/utility class - CPP Source file.

```
#include "Crc8.h"
```
Include dependency graph for Crc8.cpp:



6.1.1 Detailed Description

8 bit CRC helper/utility class - CPP Source file.

Author

J. F. Fitter jfitter@eagleairaust.com.au

Version

1.0

Date

2014-2017

Copyright

Copyright (c) 2017 John Fitter. All right reserved.

License

Definition in file Crc8.cpp.

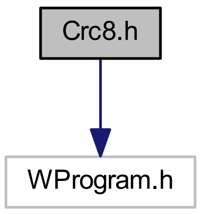
6.2 Crc8.cpp

```
/*************************************************************************************************//**
 *  \brief      8 bit CRC helper/utility class - CPP Source file.
 *  \file       CRC8.CPP
 *  \author     J. F. Fitter <jfitter@eagleairaust.com.au>
 *  \version    1.0
 *  \date       2014-2017
 *  \copyright  Copyright (c) 2017 John Fitter.  All right reserved.
 *
 *  \par        License
 *              This program is free software; you can redistribute it and/or modify it under
 *              the terms of the GNU Lesser General Public License as published by the Free
 *              Software Foundation; either version 2.1 of the License, or (at your option)
 *              any later version.
 *  \par
 *              This Program is distributed in the hope that it will be useful, but WITHOUT ANY
 *              WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
 *              PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details
 *              at http://www.gnu.org/copyleft/gpl.html
 *  \par
 *              You should have received a copy of the GNU Lesser General Public License along
 *              with this library; if not, write to the Free Software Foundation, Inc.,
 *              51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
 *
 *//***************************************************************************************/

#include "Crc8.h"

/*****************************************************************************************/
/*  CRC8 helper class functions.                                                        */
/*****************************************************************************************/

/**
 *  \brief            CRC8 class constructor.
 *  \param [in] poly  8 bit CRC polynomial to use.
 */
CRC8::CRC8(uint8_t poly) {crc8Start(poly);}

/**
 *  \brief            Return the current value of the CRC.
 *  \return           8 bit CRC current value.
 */
uint8_t CRC8::crc8(void) {return _crc;}

/**
 *  \brief            Update the current value of the CRC.
 *  \param [in] data  New 8 bit data to be added to the CRC.
 *  \return           8 bit CRC current value.
 */
uint8_t CRC8::crc8(uint8_t data) {
    uint8_t i = 8;

    _crc ^= data;
    while(i--) _crc = _crc & 0x80 ? (_crc << 1) ^ _poly : _crc << 1;
    return _crc;
}

/**
 *  \brief            Initialize the CRC8 object.
 *  \param [in] poly  8 bit CRC polynomial to use.
 */
void CRC8::crc8Start(uint8_t poly) {
    _poly = poly;
    _crc = 0;
}
```

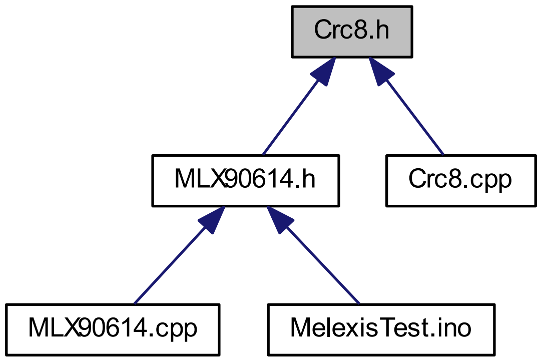
6.3 Crc8.h File Reference

8 bit CRC helper/utility class - CPP Header file.

`#include "WProgram.h"`
Include dependency graph for Crc8.h:



This graph shows which files directly or indirectly include this file:



Classes

- class CRC8

Macros

- #define CRC8_DEFAULTPOLY 7

6.3.1 Detailed Description

8 bit CRC helper/utility class - CPP Header file.

Author

J. F. Fitter `jfitter@eagleairaust.com.au`

Version

1.0

Date

2014-2017

Copyright

Copyright (c) 2017 John Fitter. All right reserved.

License

This program is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This Program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details at `http://www.gnu.org/copyleft/gpl.html`

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Definition in file Crc8.h.

6.3.2 Macro Definition Documentation

6.3.2.1 CRC8_DEFAULTPOLY

```
#define CRC8_DEFAULTPOLY 7
```

Default CRC polynomial = X8+X2+X1+1

Definition at line 35 of file Crc8.h.

6.4 Crc8.h

```
#ifndef _CRC8_H_
#define _CRC8_H_

/*********************************************************************************************//**
 *  \brief      8 bit CRC helper/utility class - CPP Header file.
 *  \file       CRC8.H
 *  \author     J. F. Fitter <jfitter@eagleairaust.com.au>
 *  \version    1.0
 *  \date       2014-2017
 *  \copyright  Copyright (c) 2017 John Fitter.  All right reserved.
 *
 *  \par        License
 *              This program is free software; you can redistribute it and/or modify it under
 *              the terms of the GNU Lesser General Public License as published by the Free
 *              Software Foundation; either version 2.1 of the License, or (at your option)
 *              any later version.
 *  \par
 *              This Program is distributed in the hope that it will be useful, but WITHOUT ANY
 *              WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
 *              PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details
 *              at http://www.gnu.org/copyleft/gpl.html
 *  \par
 *              You should have received a copy of the GNU Lesser General Public License along
 *              with this library; if not, write to the Free Software Foundation, Inc.,
 *              51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
 *
 *//*********************************************************************************************/
```

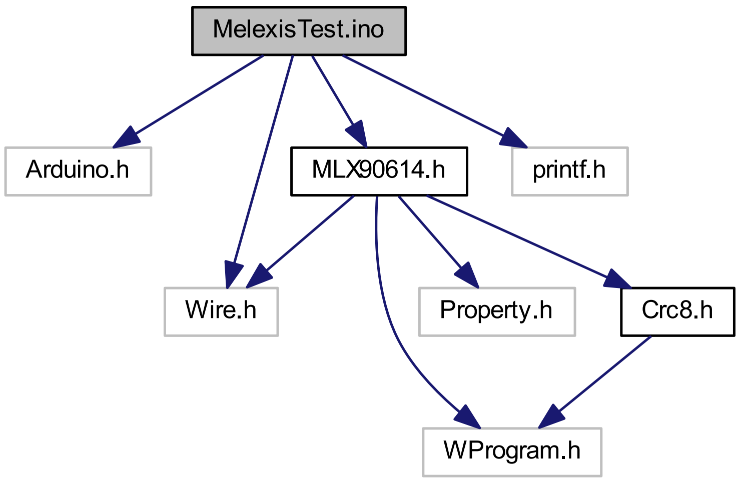
```

#if (ARDUINO >= 100)
    #include "Arduino.h"
#else
    #include "WProgram.h"
#endif

#define CRC8_DEFAULTPOLY  7  /**< Default CRC polynomial = X8+X2+X1+1 */

class CRC8 {
public:
    CRC8(uint8_t polynomial = CRC8_DEFAULTPOLY);
    uint8_t  crc8(void);
    uint8_t  crc8(uint8_t data);
    void     crc8Start(uint8_t poly);
private:
    uint8_t  _crc;
    uint8_t  _poly;
};

#endif /* _CRC8_H_ */
```

6.5 MelexisTest.ino File Reference

Melexis MCX90614BAA Test Program - Sensor test implementation.

```
#include <Arduino.h>
#include <Wire.h>
#include <MLX90614.h>
#include "printf.h"
```

Include dependency graph for MelexisTest.ino:



Classes

- struct defaultEEPromData

  EEPROM memory contents factory default values.

Functions

- void setup (void)
  Program setup.
- void loop (void)
  Main processing loop.
- void printlnTemp (double temp, char src)
  Print a line of temperature, crc, pec, and error string.
- void dumpEEProm ()
  Print a complete memory dump of the EEPROM.
- char ∗ floatToStr (char ∗str, double val)
  Utility to stringify a float.
- void printCRC (uint8_t crc, uint8_t pec)
  Just print the crc and pec.
- void printErrStr (uint8_t err)
  Convert error flags to diagnostic strings and print.
- void setEEPromDefaults (void)
  Set EEPROM memory contents to factory default values.

Variables

- MLX90614 **mlx** = MLX90614(MLX90614_BROADCASTADDR)
- const struct defaultEEPromData **eDat** [ ]

6.5.1 Detailed Description

Melexis MCX90614BAA Test Program - Sensor test implementation.

Arduino test implementation of Melexis MCX90614 PIR temperature sensor driver.

Note

THIS IS ONLY A PARTIAL RELEASE. THIS DEVICE CLASS IS CURRENTLY UNDERGOING ACTIVE D↩ EVELOPMENT AND IS STILL MISSING SOME IMPORTANT FEATURES. PLEASE KEEP THIS IN MIND IF YOU DECIDE TO USE THIS PARTICULAR CODE FOR ANYTHING.

Author

J. F. Fitter `jfitter@eagleairaust.com.au`

Version

1.0

Date

2014-2017

Copyright

License

Definition in file MelexisTest.ino.

6.5.2 Function Documentation

6.5.2.1 floatToStr()

```
char* floatToStr (
            char * str,
            double val )
```

Utility to stringify a float.

Parameters

in	*str*	String to receive converted result
in	*val*	Float value

Returns

Float as string

Definition at line 128 of file MelexisTest.ino.

```
00128                                         {
00129 
00130     sprintf(str, "%4d.%02u", int(val), int(val * 100) % 100);
00131     return str;
00132 }
```

6.5.2.2 printCRC()

```
void printCRC (
              uint8_t crc,
              uint8_t pec )
```

Just print the crc and pec.

Parameters

in	*crc*	CRC
in	*pec*	PEC

Definition at line 139 of file MelexisTest.ino.

```
{printf("crc=%02Xh pec=%02Xh", crc, pec);}
```

6.5.2.3 printErrStr()

```
void printErrStr (
              uint8_t err )
```

Convert error flags to diagnostic strings and print.

Parameters

in	*err*	Error flags

Definition at line 145 of file MelexisTest.ino.

```
                                   {

    Serial.print(F("  *** "));
    if(err == MLX90614_NORWERROR) Serial.print(F("RW Success"));
    else {
        Serial.print(F("Errors: "));
        if(err &  MLX90614_DATATOOLONG) Serial.print(F("Data too long / "));
        if(err &  MLX90614_TXADDRNACK)  Serial.print(F("TX addr NACK / "));
        if(err &  MLX90614_TXDATANACK)  Serial.print(F("TX data NACK / "));
        if(err &  MLX90614_TXOTHER)     Serial.print(F("Unknown / "));
        if(err &  MLX90614_RXCRC)       Serial.print(F("RX CRC / "));
        if(err &  MLX90614_INVALIDATA)  Serial.print(F("Invalid data / "));
        if(err &  MLX90614_EECORRUPT)   Serial.print(F("EEPROM / "));
        if(err &  MLX90614_RFLGERR)     Serial.print(F("RFlags / "));
    }
}
```

6.5.2.4 printlnTemp()

```
void printlnTemp (
            double temp,
            char src )
```

Print a line of temperature, crc, pec, and error string.

Parameters

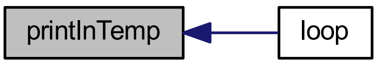
in	*temp*	Temperature
in	*src*	Temperature source

Definition at line 92 of file MelexisTest.ino.

Referenced by loop().

```
                                              {
    char str[20];

    if(mlx.rwError) Serial.print(F("No valid temperatures                              "));
    else {
        if(src == 'A') Serial.print(F("Ambient temperature"));
        else Serial.print(F("Object  temperature"));
        printf(" = %sK ", floatToStr(str, temp));
        printf("%sC ",    floatToStr(str, mlx.convKtoC(temp)));
        printf("%sF    ", floatToStr(str, mlx.convCtoF(mlx.
  convKtoC(temp))));
    }
    printCRC(mlx.crc8, mlx.pec);
    printErrStr(mlx.rwError);
    Serial.println("");
}
```

Here is the caller graph for this function:



6.5.2.5 setEEPromDefaults()

```
void setEEPromDefaults (
              void   )
```

Set EEPROM memory contents to factory default values.

Remarks

A device with default adress must not be on the bus.
`Only user allowed memory locations are written.`

Definition at line 177 of file MelexisTest.ino.

```
                                  {

    for(uint8_t i = 0; i < sizeof(eDat)/sizeof(defaultEEPromData),
        !mlx.rwError; i++) {
        mlx.writeEEProm(eDat[i].address, eDat[i].data);
    }
}
```

6.5.3 Variable Documentation

6.5.3.1 eDat

```
const struct defaultEEPromData eDat[]
```

Initial value:

```
=  {{0x20, 0x9993}, {0x21, 0x62E3}, {0x22, 0x0201},
            {0x23, 0xF71C}, {0x24, 0xFFFF}, {0x25, 0x9FB4},
            {0x2E, 0xBE5A}, {0x2F, 0x0000}, {0x39, 0x0000}}
```

6.6 MelexisTest.ino

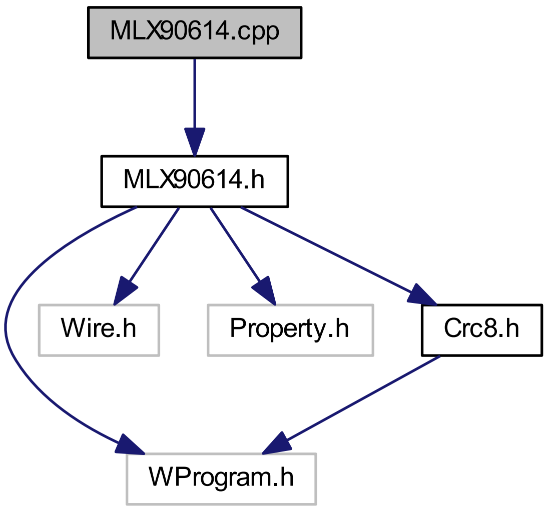
```
/***********************************************************************************************//**
 *  \brief      Melexis MCX90614BAA Test Program - Sensor test implementation.
 *  \details    Arduino test implementation of Melexis MCX90614 PIR temperature sensor driver.
 *
 *  \note       THIS IS ONLY A PARTIAL RELEASE. THIS DEVICE CLASS IS CURRENTLY UNDERGOING
 *              ACTIVE DEVELOPMENT AND IS STILL MISSING SOME IMPORTANT FEATURES. PLEASE KEEP
 *              THIS IN MIND IF YOU DECIDE TO USE THIS PARTICULAR CODE FOR ANYTHING.
 *
 *  \file       MelexisTest.ino
 *  \author     J. F. Fitter <jfitter@eagleairaust.com.au>
 *  \version    1.0
 *  \date       2014-2017
 *  \copyright  Copyright (c) 2017 John Fitter.  All right reserved.
 *
 *  \par        License
 *              This program is free software; you can redistribute it and/or modify it under
 *              the terms of the GNU Lesser General Public License as published by the Free
 *              Software Foundation; either version 2.1 of the License, or (at your option)
 *              any later version.
 *  \par
 *              This Program is distributed in the hope that it will be useful, but WITHOUT ANY
 *              WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
 *              PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details
 *              at http://www.gnu.org/copyleft/gpl.html
 *  \par
 *              You should have received a copy of the GNU Lesser General Public License along
 *              with this library; if not, write to the Free Software Foundation, Inc.,
 *              51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
 *
 *//***********************************************************************************************/

#define MELEXISTEST_C
#define __STDC_LIMIT_MACROS
#define __STDC_CONSTANT_MACROS

#include <Arduino.h>
#include <Wire.h>
#include <MLX90614.h>
#include "printf.h"

MLX90614 mlx = MLX90614(MLX90614_BROADCASTADDR);      // *** must be only one device on bus ***

/**
 *  \brief  Program setup.
 */
void setup(void) {

    Wire.begin(); // library does not do this by default
    Serial.begin(115200);
    printf_begin();
    mlx.begin();

    Serial.println(F("\nMelexis MLX90614 Temperature Sensor Test Program"));
    Serial.print(F("SMBus address ="));
    printf(" %02Xh", (uint8_t)mlx.readEEProm(MLX90614_ADDR));
    Serial.print(F("  Chip ID ="));

    uint64_t id = mlx.readID();
    printf(" %04X-%04X-%04X-%04X\n\n", (uint16_t)(id >> 48), (uint16_t)(id >> 32),
                                       (uint16_t)(id >> 16), (uint16_t)id);
    dumpEEProm();
    Serial.println("");
}

/**
 *  \brief  Main processing loop.
 */
void loop(void) {
    static uint16_t smpcount = 0, errcount = 0;

    // read ambient temperature from chip and print out
    printlnTemp(mlx.readTemp(MLX90614::MLX90614_SRCA, MLX90614::MLX90614_TK), 'A');
    if(mlx.rwError) ++errcount;

    // read object temperature from source #1 and print out
    printlnTemp(mlx.readTemp(MLX90614::MLX90614_SRC01, MLX90614::MLX90614_TK), 'O');
    if(mlx.rwError) ++errcount;

    // print running total of samples and errors
    Serial.print(F("        Samples:Errors "));
    printf("%u:%u\r\n", smpcount += 2, errcount);

    // slow down to human speed
    delay(250);
}

/**
 *  \brief          Print a line of temperature, crc, pec, and error string.
 *  \param [in] temp Temperature
 *  \param [in] src  Temperature source
 */
void printlnTemp(double temp, char src) {
    char str[20];

    if(mlx.rwError) Serial.print(F("No valid temperatures                              "));
    else {
        if(src == 'A') Serial.print(F("Ambient temperature"));
        else Serial.print(F("Object  temperature"));
        printf(" = %sK ", floatToStr(str, temp));
        printf("%sC ",    floatToStr(str, mlx.convKtoC(temp)));
        printf("%sF    ", floatToStr(str, mlx.convCtoF(mlx.convKtoC(temp))));
    }
    printCRC(mlx.crc8, mlx.pec);
    printErrStr(mlx.rwError);
    Serial.println("");
}

/**
 *  \brief Print a complete memory dump of the EEPROM.
 */
void dumpEEProm() {

    Serial.println(F("EEProm Dump"));
    for(uint8_t j=0; j<8; j++) {
        for(uint8_t i=0; i<4; i++) printf("%02Xh-%04Xh    ", j*4+i, mlx.readEEProm(j*4+i));
        printCRC(mlx.crc8, mlx.pec);
        printErrStr(mlx.rwError);
        Serial.println("");
    }
}

/**
 *  \brief          Utility to stringify a float.
 *  \param [in] str String to receive converted result
 *  \param [in] val Float value
 *  \return         Float as string
 */
char* floatToStr(char *str, double val) {

    sprintf(str, "%4d.%02u", int(val), int(val * 100) % 100);
    return str;
}

/**
 *  \brief          Just print the crc and pec.
 *  \param [in] crc CRC
 *  \param [in] pec PEC
 */
void printCRC(uint8_t crc, uint8_t pec) {printf("crc=%02Xh pec=%02Xh", crc, pec);}

/**
```

```
 *  \brief          Convert error flags to diagnostic strings and print.
 *  \param [in] err Error flags
 */
void printErrStr(uint8_t err) {

    Serial.print(F("  *** "));
    if(err == MLX90614_NORWERROR) Serial.print(F("RW Success"));
    else {
        Serial.print(F("Errors: "));
        if(err &  MLX90614_DATATOOLONG) Serial.print(F("Data too long / "));
        if(err &  MLX90614_TXADDRNACK)  Serial.print(F("TX addr NACK / "));
        if(err &  MLX90614_TXDATANACK)  Serial.print(F("TX data NACK / "));
        if(err &  MLX90614_TXOTHER)     Serial.print(F("Unknown / "));
        if(err &  MLX90614_RXCRC)       Serial.print(F("RX CRC / "));
        if(err &  MLX90614_INVALIDATA)  Serial.print(F("Invalid data / "));
        if(err &  MLX90614_EECORRUPT)   Serial.print(F("EEPROM / "));
        if(err &  MLX90614_RFLGERR)     Serial.print(F("RFlags / "));
    }
}

/**
 *  \brief  EEPROM memory contents factory default values.
 */
const struct defaultEEPromData {
    uint8_t  address;
    uint16_t data;
} eDat[] =  {{0x20, 0x9993}, {0x21, 0x62E3}, {0x22, 0x0201},
             {0x23, 0xF71C}, {0x24, 0xFFFF}, {0x25, 0x9FB4},
             {0x2E, 0xBE5A}, {0x2F, 0x0000}, {0x39, 0x0000}};

/**
 *  \brief    Set EEPROM memory contents to factory default values.
 *  \remarks  A device with default adress must not be on the bus.
 *            \n<tt>Only user allowed memory locations are written.</tt>
 */
void setEEPromDefaults(void) {

    for(uint8_t i = 0; i < sizeof(eDat)/sizeof(defaultEEPromData),
        !mlx.rwError; i++) {
        mlx.writeEEProm(eDat[i].address, eDat[i].data);
    }
}

```

6.7 MLX90614.cpp File Reference

Melexis MLX90614 Family Device Driver Library - CPP Source file.

`#include "MLX90614.h"`
Include dependency graph for MLX90614.cpp:



6.7.1 Detailed Description

Melexis MLX90614 Family Device Driver Library - CPP Source file.

Details

Based on the Melexis MLX90614 Family Data Sheet 3901090614 Rev 004 09jun2008.

- The current implementation does not manage PWM (only digital data by I2C).
- Sleep mode is not implemented yet.

Note

THIS IS ONLY A PARTIAL RELEASE. THIS DEVICE CLASS IS CURRENTLY UNDERGOING ACTIVE D↩ EVELOPMENT AND IS STILL MISSING SOME IMPORTANT FEATURES. PLEASE KEEP THIS IN MIND IF YOU DECIDE TO USE THIS PARTICULAR CODE FOR ANYTHING.

Author

J. F. Fitter `jfitter@eagleairaust.com.au`

Version

1.0

Date

2014-2017

Copyright

Copyright (c) 2017 John Fitter. All right reserved.

License

Definition in file MLX90614.cpp.

6.8 MLX90614.cpp

```
/**************************************************************************************************//**
 *  \brief      Melexis MLX90614 Family Device Driver Library - CPP Source file
 *  \par
 *  \par        Details
 *              Based on the Melexis MLX90614 Family Data Sheet 3901090614 Rev 004 09jun2008.
 *  \li         The current implementation does not manage PWM (only digital data by I2C).
 *  \li         Sleep mode is not implemented yet.
 *
 *  \note       THIS IS ONLY A PARTIAL RELEASE. THIS DEVICE CLASS IS CURRENTLY UNDERGOING
 *              ACTIVE DEVELOPMENT AND IS STILL MISSING SOME IMPORTANT FEATURES. PLEASE KEEP
 *              THIS IN MIND IF YOU DECIDE TO USE THIS PARTICULAR CODE FOR ANYTHING.
 *
 *  \file       MLX90614.CPP
 *  \author     J. F. Fitter <jfitter@eagleairaust.com.au>
 *  \version    1.0
 *  \date       2014-2017
 *  \copyright  Copyright (c) 2017 John Fitter.  All right reserved.
 *
 *  \par        License
 *              This program is free software; you can redistribute it and/or modify it under
 *              the terms of the GNU Lesser General Public License as published by the Free
 *              Software Foundation; either version 2.1 of the License, or (at your option)
 *              any later version.
 *  \par
 *              This Program is distributed in the hope that it will be useful, but WITHOUT ANY
 *              WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
 *              PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details
 *              at http://www.gnu.org/copyleft/gpl.html
 *  \par
 *              You should have received a copy of the GNU Lesser General Public License along
 *              with this library; if not, write to the Free Software Foundation, Inc.,
 *              51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
 *
 *//*************************************************************************************************/

#include "MLX90614.h"

/****************************************************************************************************/
/*  MLX90614 Device class functions.                                                               */
/****************************************************************************************************/

/**
```

```
 *  \brief               MLX90614 Device class constructor.
 *  \param [in] i2caddr  Device address (default: published value).
 */
MLX90614::MLX90614(uint8_t i2caddr) {

    busAddr.Set_Class(this);
    busAddr.Set_Get(&MLX90614::getAddr);
    busAddr.Set_Set(&MLX90614::setAddr);

    rwError.Set_Class(this);
    rwError.Set_Get(&MLX90614::getRwError);

    pec.Set_Class(this);
    pec.Set_Get(&MLX90614::getPEC);

    crc8.Set_Class(this);
    crc8.Set_Get(&MLX90614::getCRC8);

    _addr = i2caddr;
    _ready = false;
}

/**
 *  \brief  Initialize the device and the i2c interface.
 */
boolean MLX90614::begin(void) {

    _rwError = _pec = _crc8 = 0;
    return _ready = true;
}

/**
 *  \brief              Return a temperature from the specified source in specified units.
 *  \remarks
 *  \li                 Temperature is stored in ram as a 16 bit absolute value to a resolution of 0.02K
 *  \li                 Linearized sensor die temperature is available as Ta (ambient).
 *  \li                 One or two object temperatures are linearized to the range -38.2C...125C
 *  \param [in] tsrc    Internal temperature source to read, default #1.
 *  \param [in] tunit   Temperature units to convert raw data to, default deg Celsius.
 *  \return             Temperature.
 */
double MLX90614::readTemp(tempSrc_t tsrc, tempUnit_t tunit) {
    double temp;

    _rwError = 0;
    switch(tsrc) {
        case MLX90614_SRC01 : temp = read16(MLX90614_TOBJ1); break;
        case MLX90614_SRC02 : temp = read16(MLX90614_TOBJ2); break;
        default : temp = read16(MLX90614_TA);
    }
    temp *= 0.02;
    switch(tunit) {
        case MLX90614_TC : return convKtoC(temp);
        case MLX90614_TF : return convKtoC(convCtoF(temp));
    }
    return temp;
}

/**
 *  \brief              Set the emissivity of the object.
 *  \remarks            The emissivity is stored as a 16 bit integer defined by the following:
 *  \n<tt>              emissivity = dec2hex[round(65535 x emiss)]</tt>
 *  \param [in] emiss   Physical emissivity value in range 0.1 ...1.0, default 1.0
 */
void MLX90614::setEmissivity(float emiss) {

    _rwError = 0;
    uint16_t e = int(emiss * 65535. + 0.5);
    if((emiss > 1.0) || (e < 6553)) _rwError |= MLX90614_INVALIDATA;
    else writeEEProm(MLX90614_EMISS, e);
}
/**
 *  \brief              Get the emissivity of the object.
 *  \remarks            The emissivity is stored as a 16 bit integer defined by the following:
 *  \n<tt>              emissivity = dec2hex[round(65535 x emiss)]</tt>
 *  \return             Physical emissivity value in range 0.1 ...1.0
 */
float MLX90614::getEmissivity(void) {

    _rwError = 0;
    uint16_t emiss = readEEProm(MLX90614_EMISS);
    if(_rwError) return (float)1.0;
    return (float)emiss / 65535.0;
}

/**
 *  \brief             Set the coefficients of the IIR digital filter.
 *  \remarks          The IIR digital filter coefficients are set by the LS 3 bits of ConfigRegister1
 *  \n                The value of the coefficients is set as follows:
 *  \n <tt> \verbatim
csb = 0   a1 = 0.5    a2 = 0.5
      1        0.25        0.75
      2        0.167       0.833
      3        0.125       0.875
      4        1           0 (IIR bypassed)
      5        0.8         0.2
      6        0.67        0.33
      7        0.57        0.43 \endverbatim </tt>
 *  \param [in] csb   See page 12 of datasheet. Range 0...7, default = 4 (IIR bypassed)
 */
void MLX90614::setIIRcoeff(uint8_t csb) {

    _rwError = 0;

    // Ensure legal range by clearing all but the LS 3 bits.
    csb &= 7;

    // Get the current value of ConfigRegister1
    uint16_t reg = readEEProm(MLX90614_CONFIG);

    // Clear bits 2:0, mask in the new value, then write it back.
    if(!_rwError) {
        reg &= 0xfff8;
        reg |= (uint16_t)csb;
        writeEEProm(MLX90614_CONFIG, reg);
    }
}

/**
 *  \brief            Get the coefficients of the IIR digital filter.
 *  \remarks          The IIR digital filter coefficients are set by the LS 3 bits of ConfigRegister1
 *  \return           Filter coefficient table index. Range 0...7
 */
uint8_t MLX90614::getIIRcoeff(void) {

    _rwError = 0;

    // Get the current value of ConfigRegister1 bits 2:0
    uint8_t iir = readEEProm(MLX90614_CONFIG) & 7;

    if(_rwError) return 4;
    return iir;
}

/**
 *  \brief            Set the coefficients of the FIR digital filter.
 *  \remarks          The FIR digital filter coefficient N is bits 10:8 of ConfigRegister1
 *  \n                The value of N is set as follows:  <tt>N = 2 ^ (csb + 3)</tt>
    \n                The manufacturer does not recommend <tt>N < 128</tt>
 *  \param [in] csb   See page 12 of datasheet. Range 0...7, default = 7 (N = 1024)
 */
void MLX90614::setFIRcoeff(uint8_t csb) {

    _rwError = 0;

    // Ensure legal range by clearing all but the LS 3 bits.
    csb &= 7;

    // Get the current value of ConfigRegister1
    uint16_t reg = readEEProm(MLX90614_CONFIG);

    // Clear bits 10:8, mask in the new value, then write it back.
    if(!_rwError) {
        reg &= 0xf8ff;
        reg |= (uint16_t)csb << 8;
        writeEEProm(MLX90614_CONFIG, reg);
    }
}

/**
 *  \brief            Get the coefficients of the FIR digital filter.
 *  \remarks          The FIR digital filter coefficient N is bits 10:8 of ConfigRegister1
 *  \n                The value of N is set as follows:  <tt>N = 2 ^ (csb + 3)</tt>
    \n                The manufacturer does not recommend <tt>N < 128</tt>
 */
uint8_t MLX90614::getFIRcoeff(void) {

    _rwError = 0;

    // Get the current value of ConfigRegister1 bits 10:8
    uint8_t fir = (readEEProm(MLX90614_CONFIG) >> 8) & 7;

    if(_rwError) return 7;
    return fir;
```

```
}

/**
 *  \brief            Set device SMBus address.
 *  \remarks
 *  \li               Must be only device on the bus.
 *  \li               Must power cycle the device after changing address.
 *  \param [in] addr  New device address. Range 1...127
 */
void MLX90614::setAddr(uint8_t addr) {

    _rwError = 0;

    // It is assumed we do not know the existing slave address so the broadcast address is used.
    // First ensure the new address is in the legal range (1..127)
    if(addr &= 0x7f) {
        _addr = MLX90614_BROADCASTADDR;
        writeEEProm(MLX90614_ADDR, addr);

        // There will always be a r/w error using the broadcast address so we cannot respond
        // to r/w errors. We must just assume this worked.
        _addr = addr;

    } else _rwError |= MLX90614_INVALIDATA;
}

/**
 *  \brief            Return the device SMBus address.
 *  \remarks
 *  \li               Must be only device on the bus.
 *  \li               Sets the library to use the new found address.
 *  \return           Device address.
 */
uint8_t MLX90614::getAddr(void) {
    uint8_t tempAddr = _addr;

    _rwError = 0;

    // It is assumed we do not know the existing slave address so the broadcast address is used.
    // This will throw a r/w error so errors will be ignored.
    _addr = MLX90614_BROADCASTADDR;

    // Reload program copy with the existing slave address.
    _addr = lowByte(readEEProm(MLX90614_ADDR));

    return _addr;
}

/**
 *  \brief            Return a 16 bit value read from RAM or EEPROM.
 *  \param [in] cmd   Command to send (register to read from).
 *  \return           Value read from memory.
 */
uint16_t MLX90614::read16(uint8_t cmd) {
    uint16_t val;
    CRC8 crc(MLX90614_CRC8POLY);

    // Send the slave address then the command and set any error status bits returned by the write.
    Wire.beginTransmission(_addr);
    Wire.write(cmd);
    _rwError |= (1 << Wire.endTransmission(false)) >> 1;

    // Experimentally determined delay to prevent read errors (manufacturer's data sheet has
    // left something out).
    delayMicroseconds(MLX90614_XDLY);

    // Resend slave address then get the 3 returned bytes.
    Wire.requestFrom(_addr, (uint8_t)3);

    // Data is returned as 2 bytes little endian.
    val = Wire.read();
    val |= Wire.read() << 8;

    // Rread the PEC (CRC-8 of all bytes).
    _pec = Wire.read();

    // Clear r/w errors if using broadcast address.
    if(_addr == MLX90614_BROADCASTADDR) _rwError &= MLX90614_NORWERROR;

    // Build our own CRC-8 of all received bytes.
    crc.crc8(_addr << 1);
    crc.crc8(cmd);
    crc.crc8((_addr << 1) + 1);
    crc.crc8(lowByte(val));
    _crc8 = crc.crc8(highByte(val));

    // Set error status bit if CRC mismatch.
    if(_crc8 != _pec) _rwError |= MLX90614_RXCRC;

    return val;
}

/**
 *  \brief              Write a 16 bit value to memory.
 *  \param [in] cmd     Command to send (register to write to).
 *  \param [in] data    Value to write.
 */
void MLX90614::write16(uint8_t cmd, uint16_t data) {
    CRC8 crc(MLX90614_CRC8POLY);

    // Build the CRC-8 of all bytes to be sent.
    crc.crc8(_addr << 1);
    crc.crc8(cmd);
    crc.crc8(lowByte(data));
    _crc8 = crc.crc8(highByte(data));

    // Send the slave address then the command.
    Wire.beginTransmission(_addr);
    Wire.write(cmd);

    // Write the data low byte first.
    Wire.write(lowByte(data));
    Wire.write(highByte(data));

    // Then write the crc and set the r/w error status bits.
    Wire.write(_pec = _crc8);
    _rwError |= (1 << Wire.endTransmission(true)) >> 1;

    // Clear r/w errors if using broadcast address.
    if(_addr == MLX90614_BROADCASTADDR) _rwError &= MLX90614_NORWERROR;
}

/**
 *  \brief              Return a 16 bit value read from EEPROM.
 *  \param [in] addr    Register address to read from.
 *  \return             Value read from EEPROM.
 */
uint16_t MLX90614::readEEProm(uint8_t addr) {return read16(addr | 0x20);}

/**
 *  \brief              Write a 16 bit value to EEPROM after first clearing the memory.
 *  \remarks
 *  \li                 Erase and write time 5ms per manufacturer specification
 *  \li                 Manufacturer does not specify max or min erase/write times
 *  \param [in] reg     Address to write to.
 *  \param [in] data    Value to write.
 */
void MLX90614::writeEEProm(uint8_t reg, uint16_t data) {
    uint16_t val;
    reg |= 0x20;

    // Read current value, compare to the new value, and do nothing on a match or if there are
    // read errors set the error status flag only.
    val = read16(reg);
    if((val != data) && !_rwError) {

        // On any R/W errors it is assumed the memory is corrupted.
        // Clear the memory and wait Terase (per manufacturer's documentation).
        write16(reg, 0);
        delay(5);
        if(_rwError) _rwError |= MLX90614_EECORRUPT;

        // Write the data and wait Twrite (per manufacturer's documentation)
        // and set the r/w error status bits.
        write16(reg, data);
        delay(5);
        if(_rwError) _rwError |= MLX90614_EECORRUPT;
    }
}

/**
 *  \brief              Convert temperature in degrees K to degrees C.
 *  \param [in] degK    Temperature in degrees Kelvin.
 *  \return             Temperature in degrees Centigrade.
 */
double MLX90614::convKtoC(double degK) {return degK - 273.15;}

/**
 *  \brief              Convert temperature in degrees C to degrees F.
 *  \param [in] degC    Temperature in degrees Centigrade.
 *  \return             Temperature in degrees Fahrenheit.
 */
double MLX90614::convCtoF(double degC) {return (degC * 1.8) + 32.0;}

```

```
/**
 *  \brief            Retrieve the chip ID bytes.
 *  \return           Chip ID as a 64 bit word.
 */
uint64_t MLX90614::readID(void) {
    uint64_t ID = 0;

    // If we are lucky the compiler will optimise this.
    for(uint8_t i = 0; i < 4; i++) ID = (ID <<= 16) | readEEProm(MLX90614_ID1 + i);
    return ID;
}

```