

MLX90614 Device Driver

1.0

Generated by Doxygen 1.8.14

Mon Sep 16 2019 14:20:47

Contents

1	Main Page	2
2	Class Index	2
2.1	Class List	2
3	File Index	2
3.1	File List	2
4	Class Documentation	3
4.1	CRC8 Class Reference	3
4.1.1	Detailed Description	3
4.1.2	Constructor & Destructor Documentation	3
4.1.3	Member Function Documentation	4
4.2	defaultEEPromData Struct Reference	6
4.2.1	Detailed Description	6
4.3	MLX90614 Class Reference	6
4.3.1	Detailed Description	8
4.3.2	Member Enumeration Documentation	8
4.3.3	Constructor & Destructor Documentation	9
4.3.4	Member Function Documentation	9
4.3.5	Member Data Documentation	26
5	File Documentation	28
5.1	Crc8.cpp File Reference	28
5.1.1	Detailed Description	29
5.2	Crc8.cpp	29
5.3	Crc8.h File Reference	30
5.3.1	Detailed Description	31
5.3.2	Macro Definition Documentation	32
5.4	Crc8.h	32
5.5	MelexisTest.ino File Reference	33
5.5.1	Detailed Description	34
5.5.2	Function Documentation	35
5.5.3	Variable Documentation	39
5.6	MelexisTest.ino	39
5.7	MLX90614.cpp File Reference	41
5.7.1	Detailed Description	42
5.8	MLX90614.cpp	43

1 Main Page

- This is an 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.

Written by John Fitter, Eagle Air Australia p/l. and inspired by a library written by Adafruit Industries.

BSD license, all text above must be included in any redistribution

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](#) [Crc8.cpp](#) [Crc8.h](#) [property.h](#)

Place the [MLX90614](#) library folder your arduinosketchfolder/libraries/ folder. You may need to create the libraries subfolder if its your first library. Restart the IDE.

MLX90614.chm contains the documentation for the classes.

2 Class Index

2.1 Class List

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

CRC8	3
defaultEEPromData	
EEPROM memory contents factory default values	6
MLX90614	6

3 File Index

3.1 File List

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

Crc8.cpp	
8 bit CRC helper/utility class - CPP Source file	28
Crc8.h	
8 bit CRC helper/utility class - CPP Header file	30

MelexisTest.ino

Melexis MCX90614BAA Test Program - Sensor test implementation.

33

MLX90614.cppMelexis **MLX90614** Family Device Driver Library - CPP Source file

41

MLX90614.h

??

4 Class Documentation

4.1 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**

4.1.1 Detailed Description

Definition at line 37 of file **Crc8.h**.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 CRC8()

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

CRC8 class constructor.

Parameters

in	<i>poly</i>	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:



4.1.3 Member Function Documentation

4.1.3.1 [crc8\(\)](#) [1/2]

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

Return the current value of the CRC.

Returns

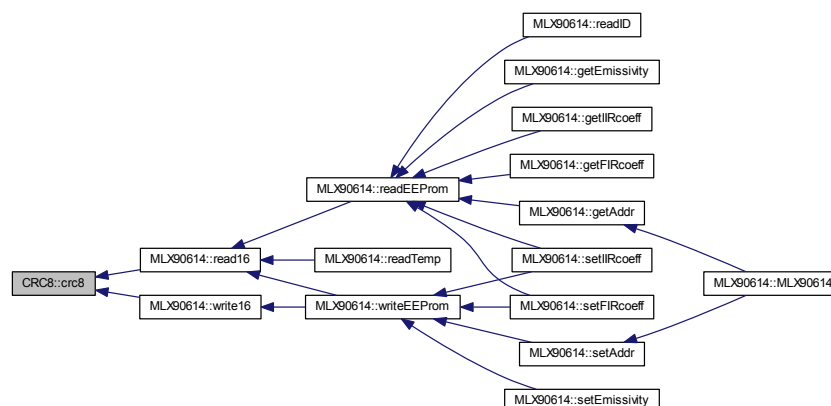
8 bit CRC current value.

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:



4.1.3.2 `crc8()` [2/2]

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

Update the current value of the CRC.

Parameters

in	<i>data</i>	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-->0) _crc = _crc & 0x80 ? (_crc << 1) ^ _poly : _crc << 1;
00054     return _crc;
00055 }
```

4.1.3.3 `crc8Start()`

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

Initialize the [CRC8](#) object.

Parameters

in	<i>poly</i>	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](#)

4.2 defaultEEPromData Struct Reference

EEPROM memory contents factory default values.

Public Attributes

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

4.2.1 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](#)

4.3 MLX90614 Class Reference

Public Types

- enum `tempUnit_t` { [MLX90614_TK](#), [MLX90614_TC](#), [MLX90614_TF](#) }
- enum `tempSrc_t` { [MLX90614_SRC_A](#), [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`

4.3.1 Detailed Description

Definition at line 104 of file [MLX90614.h](#).

4.3.2 Member Enumeration Documentation

4.3.2.1 `tempSrc_t`

enum [MLX90614::tempSrc_t](#)

Enumerations for temperature measurement source.

Enumerator

<code>MLX90614_SRC_A</code>	Chip (ambient) sensor
<code>MLX90614_SRC01</code>	IR source #1
<code>MLX90614_SRC02</code>	IR source #2

Definition at line 134 of file [MLX90614.h](#).

```
00140         :  
00141         boolean _ready;
```

4.3.2.2 `tempUnit_t`

enum [MLX90614::tempUnit_t](#)

Enumerations for temperature units.

Enumerator

<code>MLX90614_TK</code>	degrees Kelvin
<code>MLX90614_TC</code>	degrees Centigrade
<code>MLX90614_TF</code>	degrees Fahrenheit

Definition at line 129 of file [MLX90614.h](#).

```

00131         {MLX90614_SRC0,          /**< Chip (ambient) sensor */
00132         MLX90614_SRC01,        /**< IR source #1 */

```

4.3.3 Constructor & Destructor Documentation

4.3.3.1 MLX90614()

```

MLX90614::MLX90614 (
    uint8_t i2caddr = MLX90614_I2CDEFAULTADDR )

```

MLX90614 Device class constructor.

Parameters

in	<i>i2caddr</i>	Device address (default: published value).
----	----------------	--

Definition at line 46 of file [MLX90614.cpp](#).

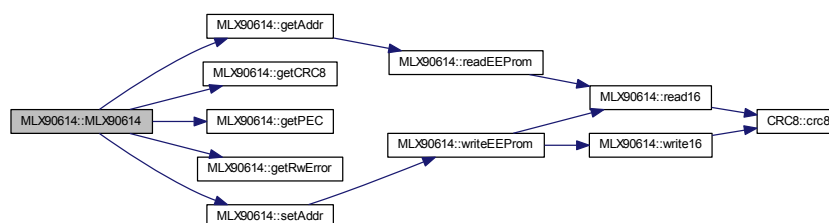
References [_addr](#), [busAddr](#), [crc8](#), [getAddr\(\)](#), [getCRC8\(\)](#), [getPEC\(\)](#), [getRwError\(\)](#), [pec](#), [rwError](#), and [setAddr\(\)](#).

```

00046         {
00047
00048         busAddr.Set_Class(this);
00049         busAddr.Set_Get (&MLX90614::getAddr);
00050         busAddr.Set_Set (&MLX90614::setAddr);
00051
00052         rwError.Set_Class(this);
00053         rwError.Set_Get (&MLX90614::getRwError);
00054
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:



4.3.4 Member Function Documentation

4.3.4.1 convCtoF()

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

Convert temperature in degrees C to degrees F.

Parameters

in	<i>degC</i>	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:



4.3.4.2 convKtoC()

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

Convert temperature in degrees K to degrees C.

Parameters

in	<i>degK</i>	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:



4.3.4.3 `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.

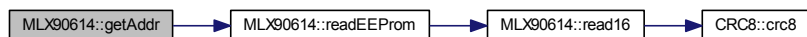
Definition at line 250 of file [MLX90614.cpp](#).

References [_addr](#), [_rwError](#), and [readEEProm\(\)](#).

Referenced by [MLX90614\(\)](#).

```
00250
00251     uint8_t tempAddr = _addr;
00252
00253     _rwError = 0;
00254
00255     // It is assumed we do not know the existing slave address so the broadcast address is used.
00256     // This will throw a r/w error so errors will be ignored.
00257     _addr = MLX90614_BROADCASTADDR;
00258
00259     // Reload program copy with the existing slave address.
00260     _addr = lowByte(readEEProm(MLX90614_ADDR));
00261
00262     return _addr;
00263 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.4.4 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:



4.3.4.5 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

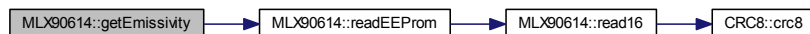
Physical emissivity value in range 0.1 ...1.0

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:



4.3.4.6 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 $N < 128$

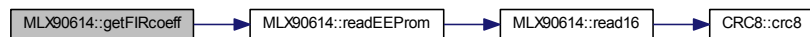
Definition at line 208 of file [MLX90614.cpp](#).

References [_rwError](#), and [readEEProm\(\)](#).

```

00208                                     {
00209
00210     _rwError = 0;
00211
00212     // Get the current value of ConfigRegister1 bits 10:8
00213     uint8_t fir = (readEEProm(MLX90614_CONFIG) >> 8) & 7;
00214
00215     if(_rwError) return 7;
00216     return fir;
00217 }
```

Here is the call graph for this function:

**4.3.4.7 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](#).

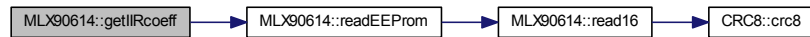
References [_rwError](#), and [readEEProm\(\)](#).

```

00166                                     {
00167
00168     _rwError = 0;
00169
00170     // Get the current value of ConfigRegister1 bits 2:0
00171     uint8_t iir = readEEProm(MLX90614_CONFIG) & 7;
00172
00173     if(_rwError) return 4;
00174     return iir;
00175 }

```

Here is the call graph for this function:



4.3.4.8 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:



4.3.4.9 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:



4.3.4.10 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\(\)](#).

```

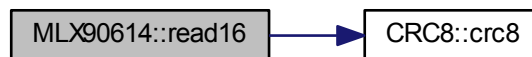
00270                                     {
00271     uint16_t val;
00272     CRC8 crc(MLX90614_CRC8POLY);
00273
00274     // Send the slave address then the command and set any error status bits returned by the write.
00275     Wire.beginTransmission(_addr);
00276     Wire.write(cmd);
00277     _rwError |= (1 << Wire.endTransmission(false)) >> 1;
00278
00279     // Experimentally determined delay to prevent read errors (manufacturer's data sheet has
00280     // left something out).
00281     delayMicroseconds(MLX90614_XDLY);
00282
00283     // Resend slave address then get the 3 returned bytes.
00284     Wire.requestFrom(_addr, (uint8_t)3);
00285
00286     // Data is returned as 2 bytes little endian.
00287     val = Wire.read();
00288     val |= Wire.read() << 8;
00289
00290     // Rread the PEC (CRC-8 of all bytes).
00291     _pec = Wire.read();
00292
00293     // Clear r/w errors if using broadcast address.
00294     if(_addr == MLX90614_BROADCASTADDR) _rwError &= MLX90614_NORWEERROR;
00295
00296     // Build our own CRC-8 of all received bytes.
00297     crc.crc8(_addr << 1);
  
```

```

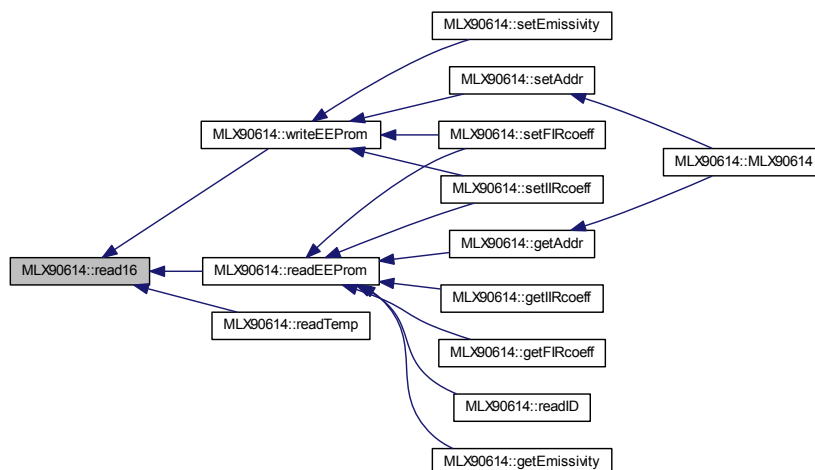
00298     crc.crc8(cmd);
00299     crc.crc8((_addr << 1) + 1);
00300     crc.crc8(lowByte(val));
00301     _crc8 = crc.crc8(highByte(val));
00302
00303     // Set error status bit if CRC mismatch.
00304     if(_crc8 != _pec) _rwError |= MLX90614_RXCRC;
00305
00306     return val;
00307 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.4.11 readEEProm()

```

uint16_t MLX90614::readEEProm (
    uint8_t addr )

```

Return a 16 bit value read from EEPROM.

Parameters

in	<i>addr</i>	Register address to read from.
----	-------------	--------------------------------

Returns

Value read from EEPROM.

Definition at line 344 of file [MLX90614.cpp](#).

References [read16\(\)](#).

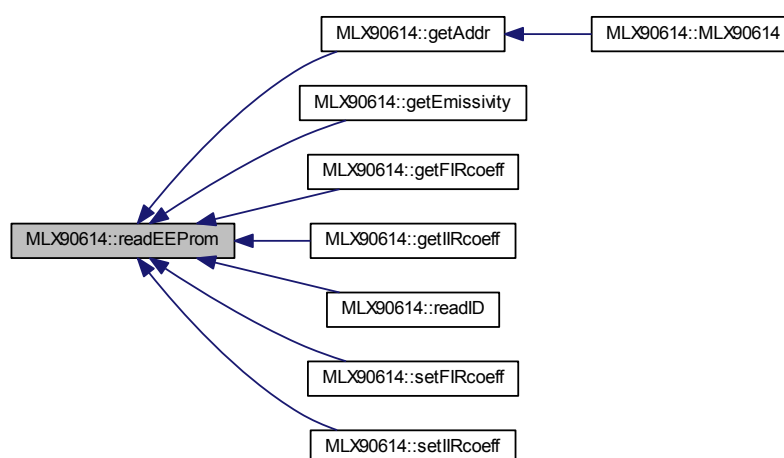
Referenced by [getAddr\(\)](#), [getEmissivity\(\)](#), [getFIRcoeff\(\)](#), [getIIRcoeff\(\)](#), [readID\(\)](#), [setFIRcoeff\(\)](#), and [setIIRcoeff\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:

**4.3.4.12 readID()**

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

Retrieve the chip ID bytes.

Chip ID getter

Returns

Chip ID as a 64 bit word.

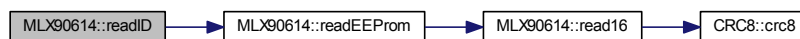
Definition at line 395 of file [MLX90614.cpp](#).

References [readEEProm\(\)](#).

```

00395                                     {
00396     uint64_t ID = 0;
00397
00398     // If we are lucky the compiler will optimise this.
00399     for(uint8_t i = 0; i < 4; i++) ID = (ID <= 16) | readEEProm(MLX90614_ID1 + i);
00400     return ID;
00401 }
```

Here is the call graph for this function:

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

Returns

Temperature.

Definition at line 84 of file [MLX90614.cpp](#).

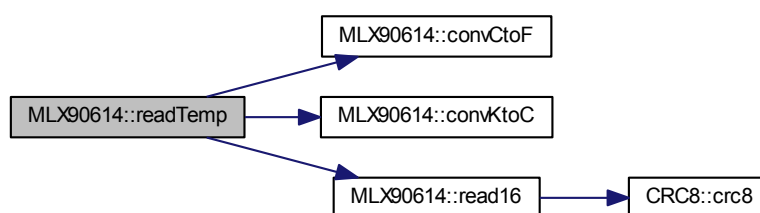
References [_rwError](#), [convCtoF\(\)](#), [convKtoC\(\)](#), [MLX90614_SRC01](#), [MLX90614_SRC02](#), [MLX90614_TC](#), [MLX90614_TF](#), and [read16\(\)](#).

```

00084                                     {
00085     double temp;
00086
00087     _rwError = 0;
00088     switch(tsrc) {
00089         case MLX90614_SRC01 : temp = read16(MLX90614_TOBJ1); break;
00090         case MLX90614_SRC02 : temp = read16(MLX90614_TOBJ2); break;
00091         default : temp = read16(MLX90614_TA);
00092     }
00093     temp *= 0.02;
00094     switch(tunit) {
00095         case MLX90614_TC : return convKtoC(temp);
00096         case MLX90614_TF : return convKtoC(convCtoF(temp));
00097     }
00098     return temp;
00099 }

```

Here is the call graph for this function:



4.3.4.14 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	<i>addr</i>	New device address. Range 1...127
----	-------------	-----------------------------------

Definition at line 226 of file [MLX90614.cpp](#).

References [_addr](#), [_rwError](#), and [writeEEProm\(\)](#).

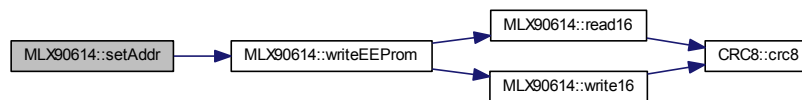
Referenced by [MLX90614\(\)](#).

```

00226                                     {
00227
00228     _rwError = 0;
00229
00230     // It is assumed we do not know the existing slave address so the broadcast address is used.
00231     // First ensure the new address is in the legal range (1..127)
00232     if(addr &= 0x7f) {
00233         _addr = MLX90614_BROADCASTADDR;
00234         writeEEProm(MLX90614_ADDR, addr);
00235
00236         // There will always be a r/w error using the broadcast address so we cannot respond
00237         // to r/w errors. We must just assume this worked.
00238         _addr = addr;
00239     } else _rwError |= MLX90614_INVALIDDATA;
00240 }
00241 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.4.15 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	<i>emiss</i>	Physical emissivity value in range 0.1 ...1.0, default 1.0
----	--------------	--

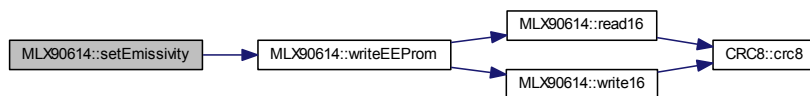
Definition at line 107 of file [MLX90614.cpp](#).

References [_rwError](#), and [writeEEProm\(\)](#).

```

00107                                     {
00108
00109     _rwError = 0;
00110     uint16_t e = int(emiss * 65535. + 0.5);
00111     if((emiss > 1.0) || (e < 6553)) _rwError |= MLX90614_INVALIDDATA;
00112     else writeEEProm(MLX90614_EMISS, e);
00113 }
```

Here is the call graph for this function:



4.3.4.16 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	<i>csb</i>	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\(\)](#).

```

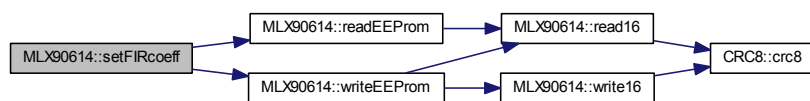
00184                                     {
```

```

00185
00186     _rwError = 0;
00187
00188     // Ensure legal range by clearing all but the LS 3 bits.
00189     csb &= 7;
00190
00191     // Get the current value of ConfigRegister1
00192     uint16_t reg = readEEProm(MLX90614_CONFIG);
00193
00194     // Clear bits 10:8, mask in the new value, then write it back.
00195     if(!_rwError) {
00196         reg &= 0xf8ff;
00197         reg |= (uint16_t)csb << 8;
00198         writeEEProm(MLX90614_CONFIG, reg);
00199     }
00200 }

```

Here is the call graph for this function:



4.3.4.17 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.

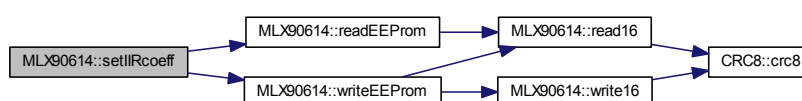
References [_rwError](#), [readEEProm\(\)](#), and [writeEEProm\(\)](#).


```

00143                                     {
00144
00145     _rwError = 0;
00146
00147     // Ensure legal range by clearing all but the LS 3 bits.
00148     csb &= 7;
00149
00150     // Get the current value of ConfigRegister1
00151     uint16_t reg = readEEProm(MLX90614_CONFIG);
00152
00153     // Clear bits 2:0, mask in the new value, then write it back.
00154     if(!_rwError) {
00155         reg &= 0xfff8;
00156         reg |= (uint16_t)csb;
00157         writeEEProm(MLX90614_CONFIG, reg);
00158     }
00159 }

```

Here is the call graph for this function:



4.3.4.18 write16()

```

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

```

Write a 16 bit value to memory.

Parameters

in	<i>cmd</i>	Command to send (register to write to).
in	<i>data</i>	Value to write.

Definition at line 314 of file [MLX90614.cpp](#).

References [_addr](#), [_crc8](#), [_pec](#), [_rwError](#), and [CRC8::crc8\(\)](#).

Referenced by [writeEEProm\(\)](#).

```

00314                                     {
00315     CRC8 crc(MLX90614_CRC8POLY);
00316
00317     // Build the CRC-8 of all bytes to be sent.
00318     crc.crc8(_addr << 1);
00319     crc.crc8(cmd);
00320     crc.crc8(lowByte(data));
00321     _crc8 = crc.crc8(highByte(data));
00322
00323     // Send the slave address then the command.
00324     Wire.beginTransmission(_addr);
00325     Wire.write(cmd);
00326

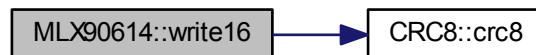
```

```

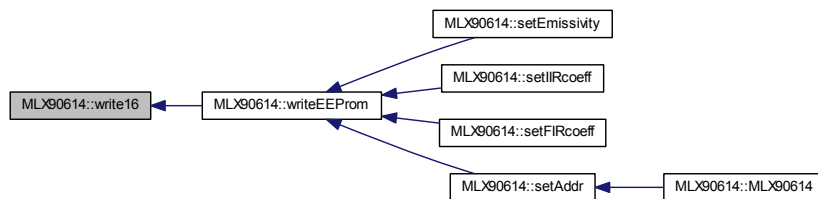
00327 // Write the data low byte first.
00328 Wire.write(lowByte(data));
00329 Wire.write(highByte(data));
00330
00331 // Then write the crc and set the r/w error status bits.
00332 Wire.write(_pec = _crc8);
00333 _rwError |= (1 << Wire.endTransmission(true)) >> 1;
00334
00335 // Clear r/w errors if using broadcast address.
00336 if(_addr == MLX90614_BROADCASTADDR) _rwError &= MLX90614_NORERROR;
00337 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.4.19 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	<i>reg</i>	Address to write to.
in	<i>data</i>	Value to write.

Definition at line 354 of file [MLX90614.cpp](#).

References [_rwError](#), [read16\(\)](#), and [write16\(\)](#).

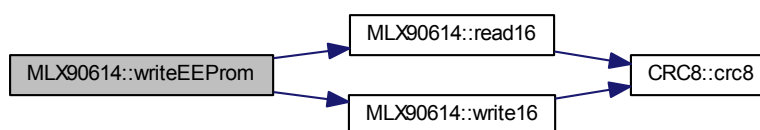
Referenced by [setAddr\(\)](#), [setEmissivity\(\)](#), [setFIRcoeff\(\)](#), and [setIIRcoeff\(\)](#).

```

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:



4.3.5 Member Data Documentation

4.3.5.1 _addr

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

Slave address

Definition at line 145 of file [MLX90614.h](#).

Referenced by [getAddr\(\)](#), [MLX90614\(\)](#), [read16\(\)](#), [setAddr\(\)](#), and [write16\(\)](#).

4.3.5.2 _crc8

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

8 bit CRC

Definition at line 147 of file [MLX90614.h](#).

Referenced by [begin\(\)](#), [read16\(\)](#), and [write16\(\)](#).

4.3.5.3 _pec

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

PEC

Definition at line 148 of file [MLX90614.h](#).

Referenced by [begin\(\)](#), [read16\(\)](#), and [write16\(\)](#).

4.3.5.4 _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\(\)](#).

4.3.5.5 busAddr

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

SMBus address property

Definition at line 123 of file [MLX90614.h](#).

Referenced by [MLX90614\(\)](#).

4.3.5.6 crc8

Property<uint8_t, [MLX90614](#)> MLX90614::crc8

8 bit CRC property

Definition at line [125](#) of file [MLX90614.h](#).

Referenced by [MLX90614\(\)](#).

4.3.5.7 pec

Property<uint8_t, [MLX90614](#)> MLX90614::pec

PEC property

Definition at line [126](#) of file [MLX90614.h](#).

Referenced by [MLX90614\(\)](#).

4.3.5.8 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](#)

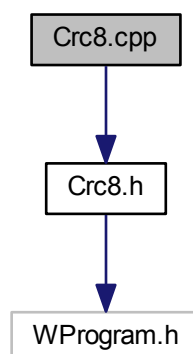
5 File Documentation

5.1 Crc8.cpp File Reference

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

```
#include "Crc8.h"
```

Include dependency graph for Crc8.cpp:



5.1.1 Detailed Description

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

Author

J. F. Fitter jfitter@eagleairaustr.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.cpp](#).

5.2 Crc8.cpp

```
00001 /*****
00002  * \brief      8 bit CRC helper/utility class - CPP Source file.
00003  * \file       CRC8.CPP
00004  * \author     J. F. Fitter <jfitter@eagleairaustr.com.au>
00005  * \version    1.0
00006  * \date       2014-2017
00007  * \copyright  Copyright (c) 2017 John Fitter. All right reserved.
00008  *
00009  * \par        License
00010  *            This program is free software; you can redistribute it and/or modify it under
00011  *            the terms of the GNU Lesser General Public License as published by the Free
00012  *            Software Foundation; either version 2.1 of the License, or (at your option)
00013  *            any later version.
00014  * \par
00015  *            This Program is distributed in the hope that it will be useful, but WITHOUT ANY
00016  *            WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
00017  *            PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details
00018  *            at http://www.gnu.org/copyleft/gpl.html
00019  * \par
00020  *            You should have received a copy of the GNU Lesser General Public License along
```

```

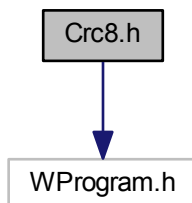
00021 *          with this library; if not, write to the Free Software Foundation, Inc.,
00022 *          51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
00023 *
00024 */*****
00025
00026 #include "Crc8.h"
00027
00028 /*****
00029 */ CRC8 helper class functions.
00030 /*****
00031
00032 /**
00033 * \brief          CRC8 class constructor.
00034 * \param [in] poly 8 bit CRC polynomial to use.
00035 */
00036 CRC8::CRC8(uint8_t poly) {crc8Start(poly);}
00037
00038 /**
00039 * \brief          Return the current value of the CRC.
00040 * \return         8 bit CRC current value.
00041 */
00042 uint8_t CRC8::crc8(void) {return _crc;}
00043
00044 /**
00045 * \brief          Update the current value of the CRC.
00046 * \param [in] data New 8 bit data to be added to the CRC.
00047 * \return         8 bit CRC current value.
00048 */
00049 uint8_t CRC8::crc8(uint8_t data) {
00050     uint8_t i = 8;
00051
00052     _crc ^= data;
00053     while(i-->0) _crc = _crc & 0x80 ? (_crc << 1) ^ _poly : _crc << 1;
00054     return _crc;
00055 }
00056
00057 /**
00058 * \brief          Initialize the CRC8 object.
00059 * \param [in] poly 8 bit CRC polynomial to use.
00060 */
00061 void CRC8::crc8Start(uint8_t poly) {
00062     _poly = poly;
00063     _crc = 0;
00064 }
00065

```

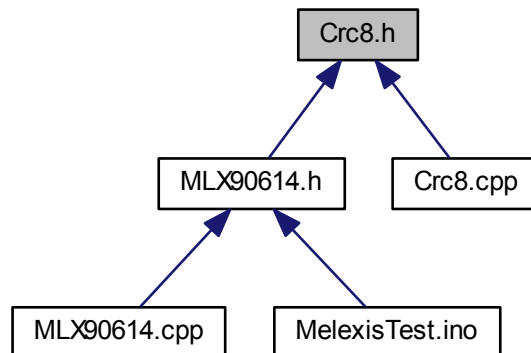
5.3 Crc8.h File Reference

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

5.3.1 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](#).

5.3.2 Macro Definition Documentation

5.3.2.1 CRC8_DEFAULTPOLY

```
#define CRC8_DEFAULTPOLY 7
```

Default CRC polynomial = X^8+X^2+X+1

Definition at line 35 of file [Crc8.h](#).

5.4 Crc8.h

```
00001 #ifndef _CRC8_H_
00002 #define _CRC8_H_
00003
00004 /*****
00005  * \brief      8 bit CRC helper/utility class - CPP Header file.
00006  * \file       CRC8.H
00007  * \author     J. F. Fitter <jfitter@eagleairust.com.au>
00008  * \version    1.0
00009  * \date       2014-2017
00010  * \copyright  Copyright (c) 2017 John Fitter. All right reserved.
00011  *
00012  * \par        License
00013  * This program is free software; you can redistribute it and/or modify it under
00014  * the terms of the GNU Lesser General Public License as published by the Free
00015  * Software Foundation; either version 2.1 of the License, or (at your option)
00016  * any later version.
00017  * \par        This Program is distributed in the hope that it will be useful, but WITHOUT ANY
00018  * WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
00019  * PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details
00020  * at http://www.gnu.org/copyleft/gpl.html
00021  * \par        You should have received a copy of the GNU Lesser General Public License along
00022  * with this library; if not, write to the Free Software Foundation, Inc.,
00023  * 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
00024  *
00025  *
00026  *
00027  */*****/
```

```

00028
00029 #if (ARDUINO >= 100)
00030     #include "Arduino.h"
00031 #else
00032     #include "WProgram.h"
00033 #endif
00034
00035 #define CRC8_DEFAULTPOLY 7 /**< Default CRC polynomial = X8+X2+X1+1 */
00036
00037 class CRC8 {
00038 public:
00039     CRC8(uint8_t polynomial = CRC8_DEFAULTPOLY);
00040     uint8_t  crc8(void);
00041     uint8_t  crc8(uint8_t data);
00042     void     crc8Start(uint8_t poly);
00043 private:
00044     uint8_t  _crc;
00045     uint8_t  _poly;
00046 };
00047
00048 #endif /* _CRC8_H_ */

```

5.5 MelexisTest.ino File Reference

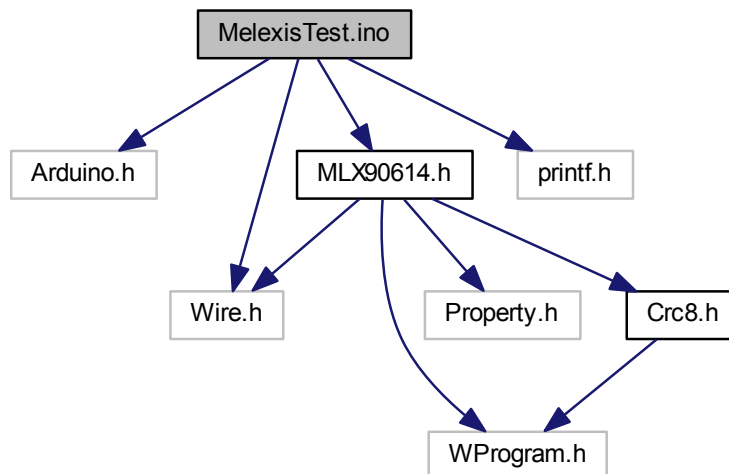
Melexis MCX90614BAA Test Program - Sensor test implementation.

```

#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` []

5.5.1 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 DEVELOPMENT 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@eagleairaustr.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 [MelexisTest.ino](#).

5.5.2 Function Documentation**5.5.2.1 floatToStr()**

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

Utility to stringify a float.

Parameters

in	<i>str</i>	String to receive converted result
in	<i>val</i>	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 }
```

5.5.2.2 printCRC()

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

Just print the crc and pec.

Parameters

in	<i>crc</i>	CRC
in	<i>pec</i>	PEC

Definition at line 139 of file [MelexisTest.ino](#).

```
00139 {printf("crc=%02Xh pec=%02Xh", crc, pec);}
```

5.5.2.3 printErrStr()

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

Convert error flags to diagnostic strings and print.

Parameters

in	<i>err</i>	Error flags
----	------------	-------------

Definition at line 145 of file [MelexisTest.ino](#).

```
00145                                     {
00146
00147     Serial.print(F(" *** "));
00148     if(err == MLX90614_NORWERROR) Serial.print(F("RW Success"));
00149     else {
00150         Serial.print(F("Errors: "));
00151         if(err & MLX90614_DATATOOLONG) Serial.print(F("Data too long / "));
00152         if(err & MLX90614_TXADDRNACK) Serial.print(F("TX addr NACK / "));
00153         if(err & MLX90614_TXDATANACK) Serial.print(F("TX data NACK / "));
00154         if(err & MLX90614_TXOTHER) Serial.print(F("Unknown / "));
00155         if(err & MLX90614_RXCRC) Serial.print(F("RX CRC / "));
00156         if(err & MLX90614_INVALIDDATA) Serial.print(F("Invalid data / "));
00157         if(err & MLX90614_EECORRUPT) Serial.print(F("EEPROM / "));
00158         if(err & MLX90614_RFLGERR) Serial.print(F("RFlags / "));
00159     }
00160 }
```

5.5.2.4 printInTemp()

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

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

Parameters

in	<i>temp</i>	Temperature
in	<i>src</i>	Temperature source

Definition at line 92 of file [MelexisTest.ino](#).

Referenced by [loop\(\)](#).

```

00092                                     {
00093     char str[20];
00094
00095     if (mlx.rwError) Serial.print(F("No valid temperatures          "));
00096     else {
00097         if (src == 'A') Serial.print(F("Ambient temperature"));
00098         else Serial.print(F("Object temperature"));
00099         printf(" = %sK ", floatToStr(str, temp));
00100         printf("%sC ", floatToStr(str, mlx.convKtoC(temp)));
00101         printf("%sF ", floatToStr(str, mlx.convCtoF(mlx.
00102     convKtoC(temp))));
00103     }
00104     printCRC(mlx.crc8, mlx.pec);
00105     printErrStr(mlx.rwError);
00106     Serial.println("");
00107 }

```

Here is the caller graph for this function:



5.5.2.5 setEEPromDefaults()

```

void setEEPromDefaults (
    void )

```

Set EEPROM memory contents to factory default values.

Remarks

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

Definition at line 177 of file [MelexisTest.ino](#).

```

00177                                     {
00178
00179     for (uint8_t i = 0; i < sizeof(eDat)/sizeof(defaultEEPromData),
00180         !mlx.rwError; i++) {
00181         mlx.writeEEProm(eDat[i].address, eDat[i].data);
00182     }
00183 }

```

5.5.3 Variable Documentation

5.5.3.1 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}}
```

5.6 MelexisTest.ino

```
00001 /*****
00002 * \brief      Melexis MCX90614BAA Test Program - Sensor test implementation.
00003 * \details    Arduino test implementation of Melexis MCX90614 PIR temperature sensor driver.
00004 *
00005 * \note       THIS IS ONLY A PARTIAL RELEASE. THIS DEVICE CLASS IS CURRENTLY UNDERGOING
00006 *             ACTIVE DEVELOPMENT AND IS STILL MISSING SOME IMPORTANT FEATURES. PLEASE KEEP
00007 *             THIS IN MIND IF YOU DECIDE TO USE THIS PARTICULAR CODE FOR ANYTHING.
00008 *
00009 * \file       MelexisTest.ino
00010 * \author     J. F. Fitter <jfitter@eagleairust.com.au>
00011 * \version    1.0
00012 * \date       2014-2017
00013 * \copyright   Copyright (c) 2017 John Fitter. All right reserved.
00014 *
00015 * \par        License
00016 *             This program is free software; you can redistribute it and/or modify it under
00017 *             the terms of the GNU Lesser General Public License as published by the Free
00018 *             Software Foundation; either version 2.1 of the License, or (at your option)
00019 *             any later version.
00020 * \par
00021 *             This Program is distributed in the hope that it will be useful, but WITHOUT ANY
00022 *             WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
00023 *             PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details
00024 *             at http://www.gnu.org/copyleft/gpl.html
00025 * \par
00026 *             You should have received a copy of the GNU Lesser General Public License along
00027 *             with this library; if not, write to the Free Software Foundation, Inc.,
00028 *             51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
00029 *
00030 */**
00031
00032 #define MELEXISTEST_C
00033 #define __STDC_LIMIT_MACROS
00034 #define __STDC_CONSTANT_MACROS
00035
00036 #include <Arduino.h>
00037 #include <Wire.h>
00038 #include <MLX90614.h>
00039 #include "printf.h"
00040
00041 MLX90614 mlx = MLX90614(MLX90614_BROADCASTADDR); // *** must be only one device on bus
00042 ***
00043 /**
00044 * \brief      Program setup.
00045 */
00046 void setup(void) {
00047
00048     Wire.begin(); // library does not do this by default
00049     Serial.begin(115200);
00050     printf_begin();
00051     mlx.begin();
00052
00053     Serial.println(F("\nMelexis MLX90614 Temperature Sensor Test Program"));
00054     Serial.print(F("SMBus address ="));
00055     printf(F(" %02Xh", (uint8_t)mlx.readEEProm(MLX90614_ADDR)));
00056     Serial.print(F(" Chip ID ="));
00057
00058     uint64_t id = mlx.readID();
```



```

00059     printf(" %04X-%04X-%04X-%04X\n\n", (uint16_t)(id >> 48), (uint16_t)(id >> 32),
00060                                     (uint16_t)(id >> 16), (uint16_t)id);
00061     dumpEEProm();
00062     Serial.println("");
00063 }
00064
00065 /**
00066  * \brief Main processing loop.
00067  */
00068 void loop(void) {
00069     static uint16_t smpcount = 0, errcount = 0;
00070
00071     // read ambient temperature from chip and print out
00072     printlnTemp(mlx.readTemp(MLX90614::MLX90614_SRC_A,
MLX90614::MLX90614_TK), 'A');
00073     if(mlx.rwError) ++errcount;
00074
00075     // read object temperature from source #1 and print out
00076     printlnTemp(mlx.readTemp(MLX90614::MLX90614_SRC_O1,
MLX90614::MLX90614_TK), 'O');
00077     if(mlx.rwError) ++errcount;
00078
00079     // print running total of samples and errors
00080     Serial.print(F("          Samples:Errors "));
00081     printf("%u:%u\r\n", smpcount += 2, errcount);
00082
00083     // slow down to human speed
00084     delay(250);
00085 }
00086
00087 /**
00088  * \brief          Print a line of temperature, crc, pec, and error string.
00089  * \param [in] temp Temperature
00090  * \param [in] src  Temperature source
00091  */
00092 void printlnTemp(double temp, char src) {
00093     char str[20];
00094
00095     if(mlx.rwError) Serial.print(F("No valid temperatures          "));
00096     else {
00097         if(src == 'A') Serial.print(F("Ambient temperature"));
00098         else Serial.print(F("Object temperature"));
00099         printf(" = %sK ", floatToStr(str, temp));
00100         printf("%sC ", floatToStr(str, mlx.convKtoC(temp)));
00101         printf("%sF ", floatToStr(str, mlx.convCtoF(mlx.
convKtoC(temp))));
00102     }
00103     printCRC(mlx.crc8, mlx.pec);
00104     printErrStr(mlx.rwError);
00105     Serial.println("");
00106 }
00107
00108 /**
00109  * \brief Print a complete memory dump of the EEPROM.
00110  */
00111 void dumpEEProm() {
00112     Serial.println(F("EEProm Dump"));
00113     for(uint8_t j=0; j<8; j++) {
00114         for(uint8_t i=0; i<4; i++) printf("%02Xh-%04Xh      ", j*4+i, mlx.
readEEProm(j*4+i));
00115         printCRC(mlx.crc8, mlx.pec);
00116         printErrStr(mlx.rwError);
00117         Serial.println("");
00118     }
00119 }
00120 }
00121
00122 /**
00123  * \brief          Utility to stringify a float.
00124  * \param [in] str String to receive converted result
00125  * \param [in] val Float value
00126  * \return         Float as string
00127  */
00128 char* floatToStr(char *str, double val) {
00129     sprintf(str, "%4d.%02u", int(val), int(val * 100) % 100);
00130     return str;
00131 }
00132 }
00133
00134 /**
00135  * \brief          Just print the crc and pec.
00136  * \param [in] crc CRC
00137  * \param [in] pec PEC
00138  */
00139 void printCRC(uint8_t crc, uint8_t pec) {printf("crc=%02Xh pec=%02Xh", crc, pec);}
00140
00141 /**

```

```

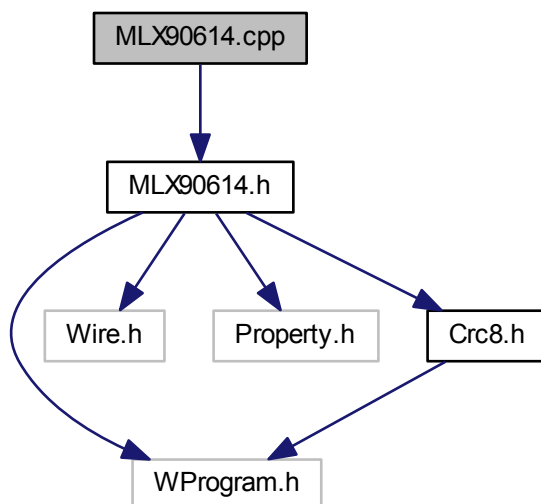
00142 * \brief          Convert error flags to diagnostic strings and print.
00143 * \param [in] err Error flags
00144 */
00145 void printErrStr(uint8_t err) {
00146     Serial.print(F(" *** "));
00147     if(err == MLX90614_NORWERROR) Serial.print(F("RW Success"));
00148     else {
00149         Serial.print(F("Errors: "));
00150         if(err & MLX90614_DATATOOLONG) Serial.print(F("Data too long / "));
00151         if(err & MLX90614_TXADDRNACK) Serial.print(F("TX addr NACK / "));
00152         if(err & MLX90614_TXDATANACK) Serial.print(F("TX data NACK / "));
00153         if(err & MLX90614_TXOTHER) Serial.print(F("Unknown / "));
00154         if(err & MLX90614_RXCRC) Serial.print(F("RX CRC / "));
00155         if(err & MLX90614_INVALIDDATA) Serial.print(F("Invalid data / "));
00156         if(err & MLX90614_EECORRUPT) Serial.print(F("EEPROM / "));
00157         if(err & MLX90614_RFLGERR) Serial.print(F("RFlags / "));
00158     }
00159 }
00160 }
00161
00162 /**
00163 * \brief  EEPROM memory contents factory default values.
00164 */
00165 const struct defaultEEPromData {
00166     uint8_t address;
00167     uint16_t data;
00168 } eDat[] = {{0x20, 0x9993}, {0x21, 0x62E3}, {0x22, 0x0201},
00169             {0x23, 0xF71C}, {0x24, 0xFFFF}, {0x25, 0x9FB4},
00170             {0x2E, 0xBE5A}, {0x2F, 0x0000}, {0x39, 0x0000}};
00171
00172 /**
00173 * \brief  Set EEPROM memory contents to factory default values.
00174 * \remarks A device with default adress must not be on the bus.
00175 *          \n<tt>Only user allowed memory locations are written.</tt>
00176 */
00177 void setEEPromDefaults(void) {
00178     for(uint8_t i = 0; i < sizeof(eDat)/sizeof(defaultEEPromData),
00179         !mlx.rwError; i++) {
00180         mlx.writeEEProm(eDat[i].address, eDat[i].data);
00181     }
00182 }
00183 }
00184

```

5.7 MLX90614.cpp File Reference

```
#include "MLX90614.h"
```

Include dependency graph for MLX90614.cpp:



5.7.1 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 DEVELOPMENT 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@eagleairaustr.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 [MLX90614.cpp](#).

5.8 MLX90614.cpp

```

00001 /*****
00002  * \brief      Melexis MLX90614 Family Device Driver Library - CPP Source file
00003  * \par
00004  * \par        Details
00005  *             Based on the Melexis MLX90614 Family Data Sheet 3901090614 Rev 004 09jun2008.
00006  * \li         The current implementation does not manage PWM (only digital data by I2C).
00007  * \li         Sleep mode is not implemented yet.
00008  *
00009  * \note       THIS IS ONLY A PARTIAL RELEASE. THIS DEVICE CLASS IS CURRENTLY UNDERGOING
00010  *             ACTIVE DEVELOPMENT AND IS STILL MISSING SOME IMPORTANT FEATURES. PLEASE KEEP
00011  *             THIS IN MIND IF YOU DECIDE TO USE THIS PARTICULAR CODE FOR ANYTHING.
00012  *
00013  * \file       MLX90614.CPP
00014  * \author     J. F. Fitter <jfitter@eagleairaut.com.au>
00015  * \version    1.0
00016  * \date       2014-2017
00017  * \copyright  Copyright (c) 2017 John Fitter. All right reserved.
00018  *
00019  * \par        License
00020  *             This program is free software; you can redistribute it and/or modify it under
00021  *             the terms of the GNU Lesser General Public License as published by the Free
00022  *             Software Foundation; either version 2.1 of the License, or (at your option)
00023  *             any later version.
00024  * \par
00025  *             This Program is distributed in the hope that it will be useful, but WITHOUT ANY
00026  *             WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
00027  *             PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details
00028  *             at http://www.gnu.org/copyleft/gpl.html
00029  * \par
00030  *             You should have received a copy of the GNU Lesser General Public License along
00031  *             with this library; if not, write to the Free Software Foundation, Inc.,
00032  *             51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
00033  *
00034  */
00035
00036 #include "MLX90614.h"
00037
00038 /*****
00039  * MLX90614 Device class functions.
00040  */
00041
00042 /**

```

```

00043 * \brief MLX90614 Device class constructor.
00044 * \param [in] i2caddr Device address (default: published value).
00045 */
00046 MLX90614::MLX90614(uint8_t i2caddr) {
00047
00048     busAddr.Set_Class(this);
00049     busAddr.Set_Get(&MLX90614::getAddr);
00050     busAddr.Set_Set(&MLX90614::setAddr);
00051
00052     rwError.Set_Class(this);
00053     rwError.Set_Get(&MLX90614::getRwError);
00054
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 }
00064
00065 /**
00066 * \brief Initialize the device and the i2c interface.
00067 */
00068 boolean MLX90614::begin(void) {
00069
00070     _rwError = _pec = _crc8 = 0;
00071     return _ready = true;
00072 }
00073
00074 /**
00075 * \brief Return a temperature from the specified source in specified units.
00076 * \remarks
00077 * \li Temperature is stored in ram as a 16 bit absolute value to a resolution of 0.02K
00078 * \li Linearized sensor die temperature is available as Ta (ambient).
00079 * \li One or two object temperatures are linearized to the range -38.2C...125C
00080 * \param [in] tsrc Internal temperature source to read, default #1.
00081 * \param [in] tunit Temperature units to convert raw data to, default deg Celsius.
00082 * \return Temperature.
00083 */
00084 double MLX90614::readTemp(tempSrc_t tsrc, tempUnit_t tunit) {
00085     double temp;
00086
00087     _rwError = 0;
00088     switch(tsrc) {
00089         case MLX90614_SRC01 : temp = read16(MLX90614_TOBJ1); break;
00090         case MLX90614_SRC02 : temp = read16(MLX90614_TOBJ2); break;
00091         default : temp = read16(MLX90614_TA);
00092     }
00093     temp *= 0.02;
00094     switch(tunit) {
00095         case MLX90614_TC : return convKtoC(temp);
00096         case MLX90614_TF : return convKtoC(convCtoF(temp));
00097     }
00098     return temp;
00099 }
00100
00101 /**
00102 * \brief Set the emissivity of the object.
00103 * \remarks The emissivity is stored as a 16 bit integer defined by the following:
00104 * \n<tt>emissivity = dec2hex[round(65535 x emiss)]</tt>
00105 * \param [in] emiss Physical emissivity value in range 0.1 ...1.0, default 1.0
00106 */
00107 void MLX90614::setEmissivity(float emiss) {
00108
00109     _rwError = 0;
00110     uint16_t e = int(emiss * 65535. + 0.5);
00111     if((emiss > 1.0) || (e < 6553)) _rwError |= MLX90614_INVALIDDATA;
00112     else writeEEProm(MLX90614_EMISS, e);
00113 }
00114 /**
00115 * \brief Get the emissivity of the object.
00116 * \remarks The emissivity is stored as a 16 bit integer defined by the following:
00117 * \n<tt>emissivity = dec2hex[round(65535 x emiss)]</tt>
00118 * \return Physical emissivity value in range 0.1 ...1.0
00119 */
00120 float MLX90614::getEmissivity(void) {
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 }
00127
00128 /**
00129 * \brief Set the coefficients of the IIR digital filter.

```

```

00130 * \remarks          The IIR digital filter coefficients are set by the LS 3 bits of ConfigRegister1
00131 * \n                The value of the coefficients is set as follows:
00132 * \n <tt> \verbatim
00133 csb = 0    a1 = 0.5    a2 = 0.5
00134         1      0.25    0.75
00135         2      0.167   0.833
00136         3      0.125   0.875
00137         4        1      0 (IIR bypassed)
00138         5        0.8    0.2
00139         6        0.67   0.33
00140         7        0.57   0.43 \endverbatim </tt>
00141 * \param [in] csb    See page 12 of datasheet. Range 0...7, default = 4 (IIR bypassed)
00142 */
00143 void MLX90614::setIIRcoeff(uint8_t csb) {
00144     _rwError = 0;
00145
00146     // Ensure legal range by clearing all but the LS 3 bits.
00147     csb &= 7;
00148
00149     // Get the current value of ConfigRegister1
00150     uint16_t reg = readEEProm(MLX90614_CONFIG);
00151
00152     // Clear bits 2:0, mask in the new value, then write it back.
00153     if(!_rwError) {
00154         reg &= 0xfff8;
00155         reg |= (uint16_t)csb;
00156         writeEEProm(MLX90614_CONFIG, reg);
00157     }
00158 }
00159 }
00160
00161 /**
00162 * \brief          Get the coefficients of the IIR digital filter.
00163 * \remarks        The IIR digital filter coefficients are set by the LS 3 bits of ConfigRegister1
00164 * \return         Filter coefficient table index. Range 0...7
00165 */
00166 uint8_t MLX90614::getIIRcoeff(void) {
00167     _rwError = 0;
00168
00169     // Get the current value of ConfigRegister1 bits 2:0
00170     uint8_t iir = readEEProm(MLX90614_CONFIG) & 7;
00171
00172     if(_rwError) return 4;
00173     return iir;
00174 }
00175 }
00176
00177 /**
00178 * \brief          Set the coefficients of the FIR digital filter.
00179 * \remarks        The FIR digital filter coefficient N is bits 10:8 of ConfigRegister1
00180 * \n              The value of N is set as follows: <tt>N = 2 ^ (csb + 3)</tt>
00181 * \n              The manufacturer does not recommend <tt>N < 128</tt>
00182 * \param [in] csb    See page 12 of datasheet. Range 0...7, default = 7 (N = 1024)
00183 */
00184 void MLX90614::setFIRcoeff(uint8_t csb) {
00185     _rwError = 0;
00186
00187     // Ensure legal range by clearing all but the LS 3 bits.
00188     csb &= 7;
00189
00190     // Get the current value of ConfigRegister1
00191     uint16_t reg = readEEProm(MLX90614_CONFIG);
00192
00193     // Clear bits 10:8, mask in the new value, then write it back.
00194     if(!_rwError) {
00195         reg &= 0xf8ff;
00196         reg |= (uint16_t)csb << 8;
00197         writeEEProm(MLX90614_CONFIG, reg);
00198     }
00199 }
00200 }
00201
00202 /**
00203 * \brief          Get the coefficients of the FIR digital filter.
00204 * \remarks        The FIR digital filter coefficient N is bits 10:8 of ConfigRegister1
00205 * \n              The value of N is set as follows: <tt>N = 2 ^ (csb + 3)</tt>
00206 * \n              The manufacturer does not recommend <tt>N < 128</tt>
00207 */
00208 uint8_t MLX90614::getFIRcoeff(void) {
00209     _rwError = 0;
00210
00211     // Get the current value of ConfigRegister1 bits 10:8
00212     uint8_t fir = (readEEProm(MLX90614_CONFIG) >> 8) & 7;
00213
00214     if(_rwError) return 7;
00215     return fir;
00216 }

```

```

00217 }
00218
00219 /**
00220 * \brief          Set device SMBus address.
00221 * \remarks
00222 * \li            Must be only device on the bus.
00223 * \li            Must power cycle the device after changing address.
00224 * \param [in] addr New device address. Range 1...127
00225 */
00226 void MLX90614::setAddr(uint8_t addr) {
00227
00228     _rWError = 0;
00229
00230     // It is assumed we do not know the existing slave address so the broadcast address is used.
00231     // First ensure the new address is in the legal range (1..127)
00232     if(addr &= 0x7f) {
00233         _addr = MLX90614_BROADCASTADDR;
00234         writeEEProm(MLX90614_ADDR, addr);
00235
00236         // There will always be a r/w error using the broadcast address so we cannot respond
00237         // to r/w errors. We must just assume this worked.
00238         _addr = addr;
00239     } else _rWError |= MLX90614_INVALIDDATA;
00241 }
00242
00243 /**
00244 * \brief          Return the device SMBus address.
00245 * \remarks
00246 * \li            Must be only device on the bus.
00247 * \li            Sets the library to use the new found address.
00248 * \return         Device address.
00249 */
00250 uint8_t MLX90614::getAddr(void) {
00251     uint8_t tempAddr = _addr;
00252
00253     _rWError = 0;
00254
00255     // It is assumed we do not know the existing slave address so the broadcast address is used.
00256     // This will throw a r/w error so errors will be ignored.
00257     _addr = MLX90614_BROADCASTADDR;
00258
00259     // Reload program copy with the existing slave address.
00260     _addr = lowByte(readEEProm(MLX90614_ADDR));
00261
00262     return _addr;
00263 }
00264
00265 /**
00266 * \brief          Return a 16 bit value read from RAM or EEPROM.
00267 * \param [in] cmd Command to send (register to read from).
00268 * \return         Value read from memory.
00269 */
00270 uint16_t MLX90614::read16(uint8_t cmd) {
00271     uint16_t val;
00272     CRC8 crc(MLX90614_CRC8POLY);
00273
00274     // Send the slave address then the command and set any error status bits returned by the write.
00275     Wire.beginTransmission(_addr);
00276     Wire.write(cmd);
00277     _rWError |= (1 << Wire.endTransmission(false)) >> 1;
00278
00279     // Experimentally determined delay to prevent read errors (manufacturer's data sheet has
00280     // left something out).
00281     delayMicroseconds(MLX90614_XDLY);
00282
00283     // Resend slave address then get the 3 returned bytes.
00284     Wire.requestFrom(_addr, (uint8_t)3);
00285
00286     // Data is returned as 2 bytes little endian.
00287     val = Wire.read();
00288     val |= Wire.read() << 8;
00289
00290     // Rread the PEC (CRC-8 of all bytes).
00291     _pec = Wire.read();
00292
00293     // Clear r/w errors if using broadcast address.
00294     if(_addr == MLX90614_BROADCASTADDR) _rWError &= MLX90614_NORWERROR;
00295
00296     // Build our own CRC-8 of all received bytes.
00297     crc.crc8(_addr << 1);
00298     crc.crc8(cmd);
00299     crc.crc8(_addr << 1 + 1);
00300     crc.crc8(lowByte(val));
00301     _crc8 = crc.crc8(highByte(val));
00302
00303     // Set error status bit if CRC mismatch.

```

```

00304     if(_crc8 != _pec) _rError |= MLX90614_RXCRC;
00305
00306     return val;
00307 }
00308
00309 /**
00310  * \brief      Write a 16 bit value to memory.
00311  * \param [in] cmd    Command to send (register to write to).
00312  * \param [in] data    Value to write.
00313  */
00314 void MLX90614::write16(uint8_t cmd, uint16_t data) {
00315     CRC8 crc(MLX90614_CRC8POLY);
00316
00317     // Build the CRC-8 of all bytes to be sent.
00318     crc.crc8(_addr << 1);
00319     crc.crc8(cmd);
00320     crc.crc8(lowByte(data));
00321     _crc8 = crc.crc8(highByte(data));
00322
00323     // Send the slave address then the command.
00324     Wire.beginTransmission(_addr);
00325     Wire.write(cmd);
00326
00327     // Write the data low byte first.
00328     Wire.write(lowByte(data));
00329     Wire.write(highByte(data));
00330
00331     // Then write the crc and set the r/w error status bits.
00332     Wire.write(_pec = _crc8);
00333     _rError |= (1 << Wire.endTransmission(true)) >> 1;
00334
00335     // Clear r/w errors if using broadcast address.
00336     if(_addr == MLX90614_BROADCASTADDR) _rError &= MLX90614_NORERROR;
00337 }
00338
00339 /**
00340  * \brief      Return a 16 bit value read from EEPROM.
00341  * \param [in] addr    Register address to read from.
00342  * \return      Value read from EEPROM.
00343  */
00344 uint16_t MLX90614::readEEProm(uint8_t addr) {return read16(addr | 0x20);}
00345
00346 /**
00347  * \brief      Write a 16 bit value to EEPROM after first clearing the memory.
00348  * \remarks
00349  * \li          Erase and write time 5ms per manufacturer specification
00350  * \li          Manufacturer does not specify max or min erase/write times
00351  * \param [in] reg    Address to write to.
00352  * \param [in] data    Value to write.
00353  */
00354 void MLX90614::writeEEProm(uint8_t reg, uint16_t data) {
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) && !_rError) {
00362
00363         // On any R/W errors it is assumed the memory is corrupted.
00364         // Clear the memory and wait Teras (per manufacturer's documentation).
00365         write16(reg, 0);
00366         delay(5);
00367         if(_rError) _rError |= 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(_rError) _rError |= MLX90614_EECORRUPT;
00374     }
00375 }
00376
00377 /**
00378  * \brief      Convert temperature in degrees K to degrees C.
00379  * \param [in] degK    Temperature in degrees Kelvin.
00380  * \return      Temperature in degrees Centigrade.
00381  */
00382 double MLX90614::convKtoC(double degK) {return degK - 273.15;}
00383
00384 /**
00385  * \brief      Convert temperature in degrees C to degrees F.
00386  * \param [in] degC    Temperature in degrees Centigrade.
00387  * \return      Temperature in degrees Fahrenheit.
00388  */
00389 double MLX90614::convCtoF(double degC) {return (degC * 1.8) + 32.0;}
00390

```



```
00391 /**
00392  * \brief      Retrieve the chip ID bytes.
00393  * \return     Chip ID as a 64 bit word.
00394  */
00395 uint64_t MLX90614::readID(void) {
00396     uint64_t ID = 0;
00397
00398     // If we are lucky the compiler will optimise this.
00399     for(uint8_t i = 0; i < 4; i++) ID = (ID <= 16) | readEEProm(MLX90614_ID1 + i);
00400     return ID;
00401 }
00402
```