Contents

1. Overview 1
2. SMBus driver 2
  2.1. *MLX90615_SMBus_Driver.cpp* 2
  2.2. *MLX90615_SWSMBus_Driver.cpp* 2
  2.3. SMBus *driver functions* 4
    *2.3.1. void MLX90615_* SMBus *Init(void)* 4
    2.3.2. void MLX90615_SMBusFreqSet(int freq) 4
    2.3.3. int MLX90615_SMBusRead(uint8_t slaveAddr, uint8_t readAddress, uint16_t *data) 5
    2.3.4. int MLX90615_SMBusWrite(uint8_t slaveAddr, uint8_t writeAddress, uint16_t data) 5
3. MLX90615 API 6
  3.1. MLX90615 configuration functions 6
    3.1.1. int MLX90615_DumpEE(uint8_t slaveAddr, uint16_t *eeData) 6
    3.1.2. int MLX90615_GetEmissivity(uint8_t slaveAddr, float *emissivity) 7
    3.1.3. int MLX90615_SetEmissivity(uint8_t slaveAddr, float value) 7
    3.1.4. int MLX90615_GetIIR(uint8_t slaveAddr, uint8_t *iir) 8
    3.1.5. int MLX90615_SetIIR(uint8_t slaveAddr, uint8_t value) 8
  3.2. MLX90615 data acquisition functions 9
    3.2.1. *int MLX90615_GetTa(uint8_t slaveAddr, float *ta)* 9
    3.2.2. *int MLX90615_GetTo(uint8_t slaveAddr, float *to)* 9
    3.2.3. *int MLX90615_GetIRdata(uint8_t slaveAddr, uint16_t *ir)* 10
  3.3. MLX90615 auxiliary functions 10
    3.3.1. *float MLX90615_TemperatureInFahrenheit(float temperature)* 10
    3.3.2. *int16_t MLX90615_ConvertIRdata(uint16_t ir)* 11
4. Revision history table 12

1. Overview

In order to use the MLX90615 driver there are 4 files that should be included in the C project:

- *MLX90615_SMBus_Driver.h* – header file containing the definitions of the SMBus related functions

- *MLX90615_SMBus_Driver.cpp or MLX90615_SWSMBus_Driver.cpp* – file containing the SMBus related functions
- *MLX90615_API.h* – header file containing the definitions of the MLX90615 specific functions
- *MLX90615_API.h* – file containing the MLX90615 specific functions

The functions return the following error codes:

- *0 – no error*
- *-1 – NACK occurred during the communication*
- *-2 – PEC discrepancy*
- *-3 – The value written in the EEPROM cell is not the same as the intended one*
- *-4 – Invalid temperature reading*
- *-6 – Invalid parameter value*

2. SMBus driver

This is the driver for the SMBus communication. The user should change this driver accordingly so that a proper SMBus communication with the MLX90615 is achieved. As the functions are being used by the MLX90615 API the functions definitions should not be changed. The SMBus standard reads LSByte first reversing the endianness of the data. Note that the driver is also responsible to reconstruct the proper endianness. If that part of the code is changed, care should be taken so that the data is properly restored. There are two SMBus drivers that could be used:

2.1. *MLX90615_SMBus_Driver.cpp*

This file should be included if the hardware SMBus in the user MCU is to be used. The user should adapt it in order to utilize the SMBus hardware module in the chosen MCU. Most MCU suppliers offer libraries with defined functions that could be used. Note that some libraries do not support repeated start condition which is required for proper communication with the MLX90615 devices.

2.2. *MLX90615_SWSMBus_Driver.cpp*

This file implements a software SMBus communication using two general purpose IOs of the MCU. The user should define the IOs (*sda* and *scl*) and ensure that the correct timing is achieved.

- Defining the IOs – SMBus data pin should be defined as an InOut pin named '*sda*',
  SMBus clock pin should be defined as an Output pin named '*scl*'

- Defining the IOs levels – in order to work properly with different hardware implementations the *scl* and *sda* levels could be defined as follows:
    - *#define LOW 0;* - low level on the line (default '0'), could be '1' if the line is inverted
    - *#define HIGH 1;* - high level on the line (default '1'), could be '0' if the line is inverted

    #define SCL_HIGH scl = HIGH; - SMBus clock high level definition

    #define SCL_LOW scl = LOW; - SMBus clock low level definition

    #define SDA_HIGH sda.input(); - SMBus data high level definition

    #define SDA_LOW sda.output(); \ - SMBus data low level definition

    sda = LOW;

    The '*sda*' pin is being switched to input for high level and to output for low level in order to allow proper work for devices that do not support open drain on the pins. This approach mimics open drain behaviour. If the device supports open drain, the definitions to set the *sda* line low and /or high could be changed.

- Setting the SMBus frequency – as this is a software implementation of the SMBus, the instruction cycle and the MCU clock affect the code execution. Therefore, in order to have the correct speed the user should modify the code so that the '*scl*' is with the desired frequency. The default implementation of the wait function is:

```
void Wait(int freqCnt)
{
        int cnt;
        for(int i = 0;i<freqCnt;i++)
        {
            cnt = cnt++;
        }
}
```

The *Wait* function could be modified in order to better trim the frequency. For coarse setting of the frequency using the dedicated function, '*freqCnt*' argument should be changed – lower value results in higher frequency.

- When writing to EEPROM a delay of at least 5ms (preferably 10ms) should be implemented. The user needs to make sure that the *void WaitEE(uint16_t ms)* function is properly trimmed. The function should be able to generate delays starting from 1ms with increments of 1ms.

2.3. SMBus *driver functions*

The SMBus driver has four main functions that ensure the proper communication between the user MCU and the MLX90615. Those functions might need some modifications by the user. However, it is important to keep the same function definitions.

*2.3.1. void MLX90615_* SMBus *Init(void)*

This function should be used to initialize the SMBus lines (sda and scl) and the SMBus hardware module if needed. The initial state of the SMBus lines should be high. The default implementation in the SMBus driver is sending a stop condition.

Example:

1. *The initialization of the SMBus should be done in the beginning of the program in order to ensure proper communication*

main.c

…definitions…

..MCU initialization

MLX90615_SMBusInit();

…

…MLX90615 communication

…User code

2.3.2. void MLX90615_SMBusFreqSet(int freq)

This function should be used to change the SMBus frequency. It has one parameter of type *int*. This parameter is used to set the frequency for a hardware SMBus module or the number of cycles in the *Wait* function in the software SMBus driver. When using SMBus hardware module, the MCU supplier provides a library with an integrated function for changing the frequency. In that case the MLX90615 SMBus driver should be changed so that it uses the library function to set the frequency. In the software SMBus driver (when using two general purpose IOs) the *SMBusFreqSet* function sets a global variable that is being used by the *Wait* function to set the number of loops. In order to set properly the frequency, the user should trim the *Wait* function so that the generated SMBus clock has the desired frequency.

Example:

1. *Setting the SMBus frequency to 50KHz when using a hardware SMBus module:*

MLX90615_SMBusFreqSet(50); *//in this case the library function provided by the MCU supplier*

// requires int value in KHz -> 50KHz

2. *Setting the SMBus frequency to 50KHz when using a software SMBus implementation:*

    MLX90615_SMBusFreqSet(160); //Depending on the instruction cycle and the clock of the MCU,

    //160 cycles in the Wait function result in 50KHz frequency of

    //the scl

2.3.3. int MLX90615_SMBusRead(uint8_t slaveAddr, uint8_t readAddress, uint16_t *data)

This function reads a word from a selected MLX90615 device memory from a given address and stores the data in the MCU memory location defined by the user. If the returned value is 0, the communication is successful, if the value is -1, NACK occurred during the communication and if the value is -2, a PEC discrepancy occurred. The function needs the following parameters:

- *uint8_t slaveAddr* – Slave address of the MLX90615 device
- *uint8_t readAddress* – address from the MLX90615 memory to be read
- *uint16_t *data* – pointer to the MCU memory location where the user wants the data to be stored

Example:

1. *Reading the configuration register EEPROM value for a MLX90615 with slave address 0x5B:*

    uint16_t cfgReg;

    status = MLX90615_SMBusRead(0x5B, 0x12, &cfgReg); //the configuration register EEPROM value //is stored in cfgReg

2. *Reading Ta data for a MLX90615 device with slave address 0x2D:*

    static uint16_t taRaw;

    status = MLX90615_SMBusRead(0x2D, 0x26, taRaw); //the raw Ta data is stored in the taRaw

2.3.4. int MLX90615_SMBusWrite(uint8_t slaveAddr, uint8_t writeAddress, uint16_t data)

This function writes a 16-bit value to a desired EEPROM address of a selected MLX90615 device. The function reads back the data after the write operation is done and returns 0 if the write was successful, -1 if NACK occurred during the communication and -3 if the data in the memory is not the same as the intended one. The following parameters are needed:

- *uint8_t slaveAddr* – Slave address of the MLX90615 device
- *uint8_t writeAddress* – address to write data to

- *uint16_t* data - Data to be written in the MLX90615 register

Example:

1. *Writing settings to the configuration register EEPROM address for a MLX90615 device with slave address 0x5B:*

    int status;

    uint16_t value;

    status = MLX90615_SMBusWrite(0x5B, 0x12, value); *//the desired settings are written to the //configuration register*

    Variable status is 0 if the write was successful.

3. MLX90615 API

This is the driver for the MLX90615 device. The user should not change this driver.

3.1. MLX90615 configuration functions

After changing the configuration values, it takes one measurement cycle (around 250ms) before the new settings are applied. Note that it could take around 500ms before valid data is available.

3.1.1. int MLX90615_DumpEE(uint8_t slaveAddr, uint16_t *eeData)

This function dumps the whole EERPOM of a MLX90615 device with a selected slave address. If the returned value is 0, the whole EEPROM dump operation is successful, if the value is -1, NACK occurred during the communication and if the value is -2, a PEC discrepancy occurred. The function needs the following parameters:

- *uint8_t slaveAddr* – Slave address of the MLX90615 device
- *uint16_t *eeData* – pointer to the MCU memory location where the EEPROM data will be stored

Example:

1. *Dumping the EEPROM from a MLX90615 device with slave address 0x5B:*

    static uint16_t eeMLX90615[16];

```
int status = 0;

status = MLX90615_DumpEE(0x5B, eeMLX90615)          // the EEPROM data is stored in eeMLX90615 array
```

3.1.2. int MLX90615_GetEmissivity(uint8_t slaveAddr, float *emissivity)

This function reads the emissivity of MLX90615 devices with a selected slave address and stores it at a MCU memory location defined by the user. If the returned value is 0, the operation is successful, if the value is -1 NACK occurred during the communication and if the value is -2 a PEC discrepancy occurred. The function needs the following parameters:

- *uint8_t slaveAddr* – Slave address of the MLX90615 device
- *float *emissivity* – pointer to the MCU memory location where the emissivity value will be stored

Example:

1. *Getting the emissivity set in a MLX90615 device with slave address 0x5B:*

```
float emissivity;

int status = 0;

status = MLX90615_GetEmissivity(0x5B, &emissivity)       // the emissivity value is stored in emissivity variable
```

3.1.3. int MLX90615_SetEmissivity(uint8_t slaveAddr, float value)

This function changes the emissivity of MLX90615 devices with a selected slave address. If the returned value is 0, the whole operation is successful, if the value is -1 NACK occurred during the communication, if the value is -2 a PEC discrepancy occurred, -3 when the value written into the EEPROM is not the same as the intended one and -6 if the emissivity parameter is not in the range [0.05:1]. The function needs the following parameters:

- *uint8_t slaveAddr* – Slave address of the MLX90615 device
- *float value* – emissivity value in the range [0.05:1]

Example:

1. *Setting emissivity = 0.95 in a MLX90615 device with slave address 0x5B:*

```
float emissivity = 0.95;

int status = 0;
```

status = MLX90615_SetEmissivity(0x5B, emissivity)   // the emissivity value is set in MLX90615 EEPROM

3.1.4. int MLX90615_GetIIR(uint8_t slaveAddr, uint8_t *iir)

This function reads the IIR setting of MLX90615 devices with a selected slave address and stores it at a MCU memory location defined by the user. If the returned value is 0, the operation is successful, if the value is -1 NACK occurred during the communication and if the value is -2 a PEC discrepancy occurred. The function needs the following parameters:

- *uint8_t slaveAddr* – Slave address of the MLX90615 device
- *uint8_t *iir* – pointer to the MCU memory location where the IIR setting value will be stored

Example:

1. *Getting the IIR setting value set from a MLX90615 device with slave address 0x5B:*

   uint8_t iir;

   int status = 0;

   status = MLX90615_GetIIR(0x5B, &iir)   // the IIR setting value is stored in iir variable

3.1.5. int MLX90615_SetIIR(uint8_t slaveAddr, uint8_t value)

This function sets the IIR setting in the EEPROM of MLX90615 devices with a selected slave address. If the returned value is 0, the operation is successful, if the value is -1 NACK occurred during the communication, if the value is -2 a PEC discrepancy occurred and if the value is -3 the value written in the EEPOM is not the same as the intended one. The function needs the following parameters:

- *uint8_t slaveAddr* – Slave address of the MLX90615 device
- *uint8_t value* – the IIR setting value in the range [0:7]

Example:

1. *Setting the IIR setting value = 5 in a MLX90615 device with slave address 0x5B:*

   uint8_t iir = 5;

   int status = 0;

   status = MLX90615_SetIIR(0x5B, iir)   // the IIR setting value is set to 5 in the device EEPROM

3.2. MLX90615 data acquisition functions

3.2.1. *int MLX90615_GetTa(uint8_t slaveAddr, float *ta)*

This function reads the current Ta measured in a given MLX90615 sensor. The result is stored at a MCU location of customer choice. If the returned value is 0, the operation is successful, if the value is -1 NACK occurred during the communication, if the value is -2 a PEC discrepancy occurred and if the value is -4 Ta value that was read is not a valid one. The function needs the following parameters:

- *uint8_t slaveAddr – Slave address of the MLX90615 device*
- *float *ta* – pointer to the MCU memory location where the Ta value will be stored

Example:

1. *Get the Ta of a MLX90615 device with slave address 0x5B that measured 23.55°C ambient temperature:*

   float ta;

   int status;

   status = MLX90615_GetTa(0x5B, &ta); *//ta = 23.55*

3.2.2. *int MLX90615_GetTo(uint8_t slaveAddr, float *to)*

This function reads the current To measured in a given MLX90615 sensor. The result is stored at a MCU location of customer choice. If the returned value is 0, the operation is successful, if the value is -1 NACK occurred during the communication, if the value is -2 a PEC discrepancy occurred and if the value is -4 To value that was read is not a valid one. The function needs the following parameters:

- *uint8_t slaveAddr – Slave address of the MLX90615 device*
- *float *to* – pointer to the MCU memory location where the To value will be stored

Example:

1. *Get the To of a MLX90615 device with slave address 0x5B that measured 35.37°C object temperature:*

   float tObj;

   int status;

status = MLX90615_GetTo(0x5B, &tObj); //tObj = 35.37

3.2.3. *int MLX90615_GetIRdata(uint8_t slaveAddr, uint16_t *ir)*

This function reads the current IRdata measured in a given MLX90615 sensor. The result is stored at a MCU location of customer choice. If the returned value is 0, the operation is successful, if the value is -1 NACK occurred during the communication and if the value is -2 a PEC discrepancy occurred. The function needs the following parameters:

- *uint8_t slaveAddr – Slave address of the MLX90615 device*
- *uint16_t *ir* – pointer to the MCU memory location where the IRdata value will be stored

Example:

1. *Get the IRdata of a MLX90615 device with slave address 0x5B:*

   uint16_t ir;

   int status;

   status = MLX90615_GetIRdata(0x5B, &ir); *//IRdata value is stored in the ir variable*

3.3. MLX90615 auxiliary functions

3.3.1. *float MLX90615_TemperatureInFahrenheit(float temperature)*

This function converts the temperature value from degrees Celsius to degrees Fahrenheit. The function needs the following parameters:

- *float temperature – temperature in °C*

Example:

1. *Convert 28.07°C to Fahrenheit:*

   float ta;

   int status;

   status = MLX90615_GetTa(0x5B, &ta); *//ta = 28.07 (in °C)*

   ta = MLX90615_TemperatureInFhrenheit(ta); *//ta = 82.53 (in F)*

3.3.2. *int16_t MLX90615_ConvertIRdata(uint16_t ir)*

This function converts the raw IRdata value from signed magnitude to 2's complement. The function needs the following parameters:

- *uint16_t ir – Raw IR data in signed magnitude*

Example:

1. *Convert IRdata value 0x001A (26dec):*

```
uint16_t rawIR;
int16_t ir;
int status;
status = MLX90615_GetIRdata(0x5B, &rawIR);          //rawIR = 0x001A
ir = MLX90615_ConvertIRdata(rawIR);                 //ir = 0x001A (26dec)
```

2. *Convert IRdata value 0x800B (-11 dec):*

```
uint16_t rawIR;
int16_t ir;
int status;
status = MLX90615_GetIRdata(0x5B, &rawIR);          //rawIR1 = 0x800B
ir = MLX90615_ConvertIRdata(rawIR);                 //ir = 0xFFF5 (-11dec)
```

4. Revision history table

26/07/2019	Initial release

Table 1