Application note
MLX90632 measurement modes

Melexis
INSPIRED ENGINEERING

Contents

1 Scope

The MLX90632 is a small size, non-contact temperature sensor. As such, it can be used in various applications which have different requirements regarding power consumption and refresh rate. In order to meet those requirements, the MLX90632 offers distinct measurement modes. This document helps understanding and implementing these measurement modes in the application.

2 Measurement modes

To control the measurement modes, register REG_CONTROL at address 0x3001 must be used. The bits that control the measurement modes are described in Table 1

Bits	Parameter	Description
11	sob	Start of Burst - starts a full table measurement when being in (sleeping) step mode
8:4	meas_select	select the type of measurement to be performed
3	soc	Start Of Conversion - starts a single measurement when being in (sleeping) step mode
2:1	mode[1:0]	defines the operating mode (step mode or continuous mode)

Table 1 – Register REG_CONTROL at address 0x3001

The temperature range modes and the power consumption modes described below can be combined in order to achieve the most suitable performance.

Regarding the temperature range, the MLX90632B## (standard accuracy) offers just one measurement mode:

- Standard accuracy temperature range – from -20°C to 200°C.

With this MLX90632 device type the standard accuracy specification is valid for the full temperature range and there is no option for extended temperature range.

The MLX90632D## (medical accuracy) offers a high accuracy in the medical range and standard accuracy in an extended temperature range. Thus, if offers two measurement modes that can be selected via the meas_select bits:

- Medical range with medical accuracy – from -20°C to 100°C. To select this mode set meas_select = 0x00.
- Extended range with standard accuracy – from -20°C to 200°C. To select this mode set meas_select = 0x11.

To control the power consumption of the sensor, both the standard accuracy and the medical accuracy devices have the following modes:

- Continuous mode – this mode is suitable for measurement applications where a very low power consumption is not a requirement. Temperature measurements are continuously ongoing and the temperature data is updated accordingly at the selected refresh rate. This is the default measurement mode of the MLX90632 sensor. In order to select it, mode[1:0] = 0b11 must be set. When in continuous mode, the temperature data is always available after the initialization time has passed, but since the device is always in active state, the typical power consumption is about 1mA.
- Step mode – this mode is suitable for measurement applications where a very tight control of the measurement is required. In order to select it, mode[1:0] = 0b10 must be set. The device is powered all the time and is in the active state. The power consumption is typically around 1mA. The device will do one measurement upon request (when soc bit is set to 1) and will wait for the next command. This

mode should only be used when a precise control of the measurement timing is required. As the thermal conditions are constantly changing, it is recommended that all the measurements from the measurement table are done with as little delay as possible. In most cases it is recommended to use the sleeping step mode instead.

- Sleeping step mode – this mode is suitable for measurement applications where a low power consumption is required. This mode also allows for a fine control of the measurements. In order to select it, mode[1:0] = 0b01 must be set. When in sleeping step mode, the device switches between two states – the sleep state with a power consumption of about 1.5µA and the active state with a power consumption of about 1mA. The normal state is the sleep state and the active state is entered only when a measurement is triggered. The duration and repetition rate of the active state defines the overall power consumption. If a measurement is never triggered, the power consumption would be equal to the sleep current - around 1.5µA. Increasing the number of measurements triggered per time period will increase the average power consumption.

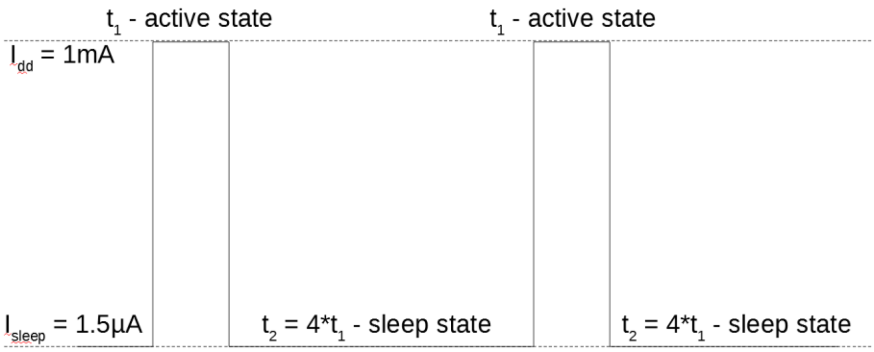

Figure 1 – Sleeping step mode power consumption example for 20% duty cycle

Duty cycle, %	Idd, µA	Description
0	1.5 (sleep)	A measurement is never triggered
20	201.2	MLX90632 is measuring for 20% of the time e.g. 1 second measurement every 5 seconds
40	400.9	MLX90632 is measuring for 40% of the time e.g. 2 seconds measurement every 5 seconds
60	600.6	MLX90632 is measuring for 60% of the time e.g. 3 seconds measurement every 5 seconds
80	800.3	MLX90632 is measuring for 80% of the time e.g. 4 seconds measurement every 5 seconds
100	1000 (Idd)	MLX90632 is measuring for 100% of the time

Table 2 – Sleeping step mode power consumption

Note: The peak current would still be the current in active state

3 Measurement flows

The different measurement modes may require different flows. In order to determine the state of the MLX90632 device, one must access the REG_STATUS register at address 0x3FFF.

Bits	Parameter	Description
10	device_busy	Flag indicating that a measurement is being executed (1 = measurement ongoing). In sleep mode, this flag is always low. In continuous mode, this flag is always high. In soc-step mode, this flag is high during one measurement. In sob-step mode, this flag is high till all measurements are finished
6:2	cycle_position	Indicates from which measurement (in the measurement table) the last written data is coming
0	new_data	Customer should set bit to 0 When a measurement is done, the bit is set to 1 Customer can readout the data and reset the bit to 0

Table 3 – Register STATUS at address 0x3FFF

Continuous standard or medical mode	Write into REG_CONTROL register
1. Select continuous mode (default)	mode[1:0] = 0b11
2. Select medical/standard mode (default)	meas_select = 0x00
3. Measurement loop	
a. Clear the new data flag	new_data = 0
b. Wait for new data	Depending on the refresh rate wait for some time. After that poll the new_data bit until it becomes 1
c. Read out all the required data	If cycle_position = 1: RAM_4, RAM_5, RAM_6 If cycle_position = 2: RAM_7, RAM_8, RAM_9
d. Do all required pre-calculations e. Calculate ambient temperature f. Calculate object temperature	

Table 4 – Continuous standard/medical mode flow

Continuous extended mode	Write into REG_CONTROL register
1. Verify that extended mode is supported	EE_VERSION value at address 0x240B bits [14:8] = 0x05
2. Select continuous mode (default)	mode[1:0] = 0b11
3. Select extended mode	meas_select = 0x11
4. Measurement loop	
a. Clear the new data flag	new_data = 0
b. Wait for new data	Depending on the refresh rate wait for some time. After that poll the new_data bit until it becomes 1. Do this until cycle_position becomes 0x13
c. Read out all the required data	RAM_52, RAM_53, RAM_54 RAM_55, RAM_56, RAM_57 RAM_58, RAM_59, RAM_60

d. Do all required pre-calculations e. Calculate ambient temperature f. Calculate object temperature	

Table 5 – Continuous extended mode flow

Step standard or medical mode	Write into REG_CONTROL register
1. Select step mode	mode[1:0] = 0b10
2. Select medical/standard mode (default)	meas_select = 0x00
3. Measurement loop	
a. Clear the new data flag	new_data = 0
b. Set soc bit to start a new measurement	soc = 1
c. Wait for new data	Depending on the refresh rate wait for some time. After that poll the new_data bit until it becomes 1
d. Clear the new data flag	new_data = 0
e. Set soc bit to start a new measurement	soc = 1
f. Wait for new data	Depending on the refresh rate wait for some time. After that poll the new_data bit until it becomes 1
g. Read out all the required data	RAM_4, RAM_5, RAM_6 RAM_7, RAM_8, RAM_9
h. Do all required pre-calculations i. Calculate ambient temperature j. Calculate object temperature	

Table 6 – Step standard/medical mode flow

Step extended mode	Write into REG_CONTROL register
1. Verify that extended mode is supported	EE_VERSION value at address 0x240B bits [14:8] = 0x05
2. Select step mode	mode[1:0] = 0b10
3. Select extended mode	meas_select = 0x11
4. Measurement loop	
a. Clear the new data flag	new_data = 0
b. Set soc bit to start a new measurement	soc = 1
c. Wait for new data	Depending on the refresh rate wait for some time. After that poll the new_data bit until it becomes 1
d. Clear the new data flag	new_data = 0
e. Set soc bit to start a new measurement	soc = 1
f. Wait for new data	Depending on the refresh rate wait for some time. After that poll the new_data bit until it becomes 1
g. Clear the new data flag	new_data = 0

h. Set soc bit to start a new measurement	soc = 1
i. Wait for new data	Depending on the refresh rate wait for some time. After that poll the new_data bit until it becomes 1
j. Read out all the required data	RAM_52, RAM_53, RAM_54 RAM_55, RAM_56, RAM_57 RAM_58, RAM_59, RAM_60
k. Do all required pre-calculations l. Calculate ambient temperature m. Calculate object temperature	

Table 7 – Step extended mode flow

Sleeping step standard or medical burst mode	Write into REG_CONTROL register
1. Select sleeping step mode	mode[1:0] = 0b01
2. Select medical/standard mode (default)	meas_select = 0x00
3. Measurement loop	
a. Start a new burst measurement	sob = 1
b. Wait for all the measurements from the table to be performed	Depending on the refresh rate wait for some time. After that poll the device_busy bit until it becomes 0
c. Read out all the required data	RAM_4, RAM_5, RAM_6 RAM_7, RAM_8, RAM_9
d. Do all required pre-calculations e. Calculate ambient temperature f. Calculate object temperature	

Table 8 – Sleeping step standard/medical mode flow

Sleeping step extended burst mode	Write into REG_CONTROL register
1. Verify that extended mode is supported	EE_VERSION value at address 0x240B bits [14:8] = 0x05
2. Select sleeping step mode (default)	mode[1:0] = 0b01
3. Select extended mode	meas_select = 0x11
4. Measurement loop	
a. Start a new burst measurement	sob = 1
b. Wait for all the measurements from the table to be performed	Depending on the refresh rate wait for some time. After that poll the device_busy bit until it becomes 0
c. Read out all the required data	RAM_52, RAM_53, RAM_54 RAM_55, RAM_56, RAM_57 RAM_58, RAM_59, RAM_60
d. Do all required pre-calculations e. Calculate ambient temperature f. Calculate object temperature	

Table 9 – Sleeping step standard/medical mode flow

4 Special considerations

- The average consumption in the burst modes can be controlled between I_{sleep} and I_{dd} by the sleep to active state ratio
- I_{dd} is the current in the active state – more information about it can be found in the datasheet – parameter I_{DD}
- I_{sleep} is the current in the sleep state – more information about it can be found in the datasheet – parameter I_{DDPR}
- The peak consumption in the burst modes would still be I_{dd} as this is the consumption in active state
- The parameters needed for the calculations that are being stored in the EEPROM could be extracted only once after power-on and stored in RAM
- It is recommended to initialize the I2C lines by generating a stop condition after power-on
- The device should be put in stepping mode before doing EEPROM operations
- Depending on the application needs, the different modes can be combined
- The typical refresh times for medical and standard modes are

EE_MEAS_1[10:8] EE_MEAS_2[10:8]	Standard meas time [ms]	Burst meas time [ms]
0	2000	4000
1	1000	2000
2	500	1000
3	250	500
4	125	250
5	62.5	125
6	31.25	62.5
7	15.625	31.25

Table 10 – Sleeping step standard/medical mode flow

- The typical refresh times for extended mode are

EE_MEAS_17[10:8] EE_MEAS_18[10:8] EE_MEAS_19[10:8]	Standard meas time [ms]	Burst meas time [ms]
0	6000	6000
1	3000	3000
2	1500	1500
3	750	750
4	375	375
5	200	200
6	100	100
7	50	50

Table 11 – Sleeping step standard/medical mode flow

5 Examples for typically used measurement modes

The measurement modes that should typically be used are:

- Continuous medical/standard mode
- Continuous extended mode
- Sleeping step medical/standard (burst) mode
- Sleeping step extended (burst) mode

All of those measurement modes can be implemented using the MLX90632 library available at
https://github.com/melexis/mlx90632-library.git
There are several common things that need to be done to use the library:

- Make sure that *BITS_PER_LONG* is properly defined. This is a MCU specific value that is being used to generate bit masks
- I2C function implementation – the I2C functions are specific for each MCU as they depend on the available hardware. The prototypes of the I2C functions are listed in *mlx90632_depends.h* file.
- Implementation of certain timing functions – as with the I2C functions, the timings depend on the available hardware and therefore need to be implemented for each MCU individually. The prototypes of the required timing functions are listed in *mlx90632_depends.h* file.
- Declare the variables that would hold the parameter values from EEPROM. One can use global or local storage.
- Declare variables to hold intermittent data for ambient_new_raw, ambient_old_raw, object_new_raw and object_old_raw
- Declare variable to store the calculated temperatures – *ambient* and *object*

5.1 Continuous medical/standard mode example

```
#include "mlx90632.h"

/* Declare and implement here functions you find in mlx90632_depends.h */

/* Declare the variables to hold the EEPROM parameters values
 * The calibration parameters could also be declared as local variables */
  int32_t PR;
  int32_t PG;
  int32_t PT;
  int32_t PO;
  int32_t Ea;
  int32_t Eb;
  int32_t Fa;
  int32_t Fb;
  int32_t Ga;
  int16_t Ha;
  int16_t Hb;
  int16_t Gb;
  int16_t Ka;

int main(void)
{
  int32_t ret = 0; /**< Variable will store return values */
  double pre_ambient; /**< Ambient pre-process */
  double pre_object; /**< Object pre-process*/
  double ambient; /**< Ambient temperature in degrees Celsius */
  double object; /**< Object temperature in degrees Celsius */

  /* ambient_new_raw,  ambient_old_raw, object_new_raw, object_old_raw */
```

```
int16_t ambient_new_raw;
int16_t ambient_old_raw;
int16_t object_new_raw;
int16_t object_old_raw;

/* Initialize the I2C lines */

/* Initialize the device and get a clean start */
ret = mlx90632_init();
if(ret == 0){
   /* Only medical/standard mode is supported */
}else if(ret == ERANGE){
   /* Extended mode is also supported */
}else{
   /* Something went wrong or invalid device */
};

/* Put the device in sleeping step mode in order to safely read the EEPROM */
ret = mlx90632_set_meas_type(MLX90632_MTYP_MEDICAL_BURST);

/* Read sensor EEPROM registers needed for calculations */

/* Set emissivity */
mlx90632_set_emissivity(1.00);

/* Set continuous medical/standard measurement mode */
if (mlx90632_get_meas_type() != MLX90632_MTYP_MEDICAL)
                ret = mlx90632_set_meas_type(MLX90632_MTYP_MEDICAL);

/* The following instructions could be looped if the mode is not being switched
 * Get raw data for ambient and object temperature calculation */
ret = mlx90632_read_temp_raw(&ambient_new_raw, &ambient_old_raw,
               &object_new_raw, &object_old_raw);
if(ret < 0)
   /* Something went wrong - abort */
   return ret;

/* Now start calculations (no more i2c accesses) */
/* Calculate ambient temperature */
ambient = mlx90632_calc_temp_ambient(ambient_new_raw, ambient_old_raw,
                  PT, PR, PG, PO, Gb);

/* Get pre-processed temperatures needed for object temperature calculation */
pre_ambient = mlx90632_preprocess_temp_ambient(ambient_new_raw,
                        ambient_old_raw, Gb);
pre_object = mlx90632_preprocess_temp_object(object_new_raw, object_old_raw,
                       ambient_new_raw, ambient_old_raw,
                       Ka);
/* Calculate object temperature */
```

```
    object = mlx90632_calc_temp_object(pre_object, pre_ambient, Ea, Eb, Ga, Fa, Fb, Ha, Hb);

   /* Do something with the temperature data */
}
```

5.2 Continuous extended mode example

```
#include "mlx90632.h"

/* Declare and implement here functions you find in mlx90632_depends.h */

/* Declare the variables to hold the EEPROM parameters values
 * The calibration parameters could also be declared as local variables */
    int32_t PR;
    int32_t PG;
    int32_t PT;
    int32_t PO;
    int32_t Ea;
    int32_t Eb;
    int32_t Fa;
    int32_t Fb;
    int32_t Ga;
    int16_t Ha;
    int16_t Hb;
    int16_t Gb;
    int16_t Ka;

int main(void)
{
    int32_t ret = 0; /**< Variable will store return values */
    double pre_ambient; /**< Ambient pre-process */
    double pre_object; /**< Object pre-process*/
    double ambient; /**< Ambient temperature in degrees Celsius */
    double object; /**< Object temperature in degrees Celsius */

    /* ambient_new_raw,  ambient_old_raw, object_new_raw, object_old_raw */
    int16_t ambient_new_raw;
    int16_t ambient_old_raw;
    int16_t object_new_raw;
    int16_t object_old_raw;

    /* Initialize the I2C lines */

    /* Initialize the device and get a clean start */
    ret = mlx90632_init();
    if(ret == 0){
       /* Only medical/standard mode is supported */
    }else if(ret == ERANGE){
       /* Extended mode is also supported */
```

```
    }else{
        /* Something went wrong or invalid device */
    };

    /* Put the device in sleeping step mode in order to safely read the EEPROM */
    ret = mlx90632_set_meas_type(MLX90632_MTYP_MEDICAL_BURST);

    /* Read sensor EEPROM registers needed for calculations */

    /* Set emissivity */
    mlx90632_set_emissivity(1.00);

    /* Set continuous extended measurement mode */
    if (mlx90632_get_meas_type() != MLX90632_MTYP_EXTENDED)
                ret = mlx90632_set_meas_type(MLX90632_MTYP_EXTENDED);

    /* The following instructions could be looped if the mode is not being switched
     * Get raw data for ambient and object temperature calculation */
    ret = mlx90632_read_temp_raw_extended(&ambient_new_raw, &ambient_old_raw,
                &object_new_raw);
    if(ret < 0)
        /* Something went wrong - abort */
        return ret;

    /* Now start calculations (no more i2c accesses) */
    /* Calculate ambient temperature */
    ambient = mlx90632_calc_temp_ambient_extended(ambient_new_raw, ambient_old_raw,
                PT, PR, PG, PO, Gb);

    /* Get pre-processed temperatures needed for object temperature calculation */
    pre_ambient = mlx90632_preprocess_temp_ambient_extended(ambient_new_raw,
                ambient_old_raw, Gb);
    pre_object = mlx90632_preprocess_temp_object_extended(object_new_raw,
                ambient_new_raw, ambient_old_raw,
                Ka);
    /* Calculate object temperature */
    object = mlx90632_calc_temp_object_extended(pre_object, pre_ambient, ambient, Ea, Eb, Ga, Fa, Fb, Ha,
Hb);

    /* Do something with the temperature data */
}
```

5.3 Burst medical/standard mode example

```
#include "mlx90632.h"

/* Declare and implement here functions you find in mlx90632_depends.h */

/* Declare the variables to hold the EEPROM parameters values
```

```
  * The calibration parameters could also be declared as local variables */
   int32_t PR;
   int32_t PG;
   int32_t PT;
   int32_t PO;
   int32_t Ea;
   int32_t Eb;
   int32_t Fa;
   int32_t Fb;
   int32_t Ga;
   int16_t Ha;
   int16_t Hb;
   int16_t Gb;
   int16_t Ka;

int main(void)
{
   int32_t ret = 0; /**< Variable will store return values */
   double pre_ambient; /**< Ambient pre-process */
   double pre_object; /**< Object pre-process*/
   double ambient; /**< Ambient temperature in degrees Celsius */
   double object; /**< Object temperature in degrees Celsius */

   /* ambient_new_raw,  ambient_old_raw, object_new_raw, object_old_raw */
   int16_t ambient_new_raw;
   int16_t ambient_old_raw;
   int16_t object_new_raw;
   int16_t object_old_raw;

   /* Initialize the I2C lines */

   /* Initialize the device and get a clean start */
   ret = mlx90632_init();
   if(ret == 0){
      /* Only medical/standard mode is supported */
   }else if(ret == ERANGE){
      /* Extended mode is also supported */
   }else{
      /* Something went wrong or invalid device */
   };

   /* Put the device in sleeping step mode in order to safely read the EEPROM */
   ret = mlx90632_set_meas_type(MLX90632_MTYP_MEDICAL_BURST);

   /* Read sensor EEPROM registers needed for calculations */

   /* Set emissivity */
   mlx90632_set_emissivity(1.00);
```

```
  /* Set burst medical/standard mode */
  if (mlx90632_get_meas_type() != MLX90632_MTYP_MEDICAL_BURST)
                         ret = mlx90632_set_meas_type(MLX90632_MTYP_MEDICAL_BURST);

  /* The following instructions could be looped if the mode is not being switched
   * Get raw data for ambient and object temperature calculation */
  ret = mlx90632_read_temp_raw_burst(&ambient_new_raw, &ambient_old_raw,
                         &object_new_raw, &object_old_raw);
  if(ret < 0)
     /* Something went wrong - abort */
     return ret;

  /* Now start calculations (no more i2c accesses) */
  /* Calculate ambient temperature */
  ambient = mlx90632_calc_temp_ambient(ambient_new_raw, ambient_old_raw,
                              PT, PR, PG, PO, Gb);

  /* Get pre-processed temperatures needed for object temperature calculation */
  pre_ambient = mlx90632_preprocess_temp_ambient(ambient_new_raw,
                                        ambient_old_raw, Gb);
  pre_object = mlx90632_preprocess_temp_object(object_new_raw, object_old_raw,
                                       ambient_new_raw, ambient_old_raw,
                                       Ka);
  /* Calculate object temperature */
  object = mlx90632_calc_temp_object(pre_object, pre_ambient, Ea, Eb, Ga, Fa, Fb, Ha, Hb);

 /* Do something with the temperature data */

 /* The measurement is done and the MLX90632 device is in sleep mode
  * Waiting here will determine the active to sleep mode ratio, which would
  * determine the average power consumption over time */
}
```

5.4 Burst extended mode example

```
#include "mlx90632.h"

/* Declare and implement here functions you find in mlx90632_depends.h */

/* Declare the variables to hold the EEPROM parameters values
 * The calibration parameters could also be declared as local variables */
  int32_t PR;
  int32_t PG;
  int32_t PT;
  int32_t PO;
  int32_t Ea;
  int32_t Eb;
  int32_t Fa;
  int32_t Fb;
```

```
    int32_t Ga;
    int16_t Ha;
    int16_t Hb;
    int16_t Gb;
    int16_t Ka;

int main(void)
{
    int32_t ret = 0; /**< Variable will store return values */
    double pre_ambient; /**< Ambient pre-process */
    double pre_object; /**< Object pre-process*/
    double ambient; /**< Ambient temperature in degrees Celsius */
    double object; /**< Object temperature in degrees Celsius */

    /* ambient_new_raw,  ambient_old_raw, object_new_raw, object_old_raw */
    int16_t ambient_new_raw;
    int16_t ambient_old_raw;
    int16_t object_new_raw;
    int16_t object_old_raw;

    /* Initialize the I2C lines */

    /* Initialize the device and get a clean start */
    ret = mlx90632_init();
    if(ret == 0){
        /* Only medical/standard mode is supported */
    }else if(ret == ERANGE){
        /* Extended mode is also supported */
    }else{
        /* Something went wrong or invalid device */
    };

    /* Put the device in sleeping step mode in order to safely read the EEPROM */
    ret = mlx90632_set_meas_type(MLX90632_MTYP_EXTENDED_BURST);

    /* Read sensor EEPROM registers needed for calculations */

    /* Set emissivity */
    mlx90632_set_emissivity(1.00);

    /* Set burst extended measurement mode */
    if (mlx90632_get_meas_type() != MLX90632_MTYP_EXTENDED_BURST)
                        ret = mlx90632_set_meas_type(MLX90632_MTYP_EXTENDED_BURST);

    /* The following instructions could be looped if the mode is not being switched
     * Get raw data for ambient and object temperature calculation */
    ret = mlx90632_read_temp_raw_extended_burst(&ambient_new_raw, &ambient_old_raw,
                       &object_new_raw);
    if(ret < 0)
```

```
        /* Something went wrong - abort */
        return ret;

    /* Now start calculations (no more i2c accesses) */
    /* Calculate ambient temperature */
    ambient = mlx90632_calc_temp_ambient_extended(ambient_new_raw, ambient_old_raw,
                                    PT, PR, PG, PO, Gb);

    /* Get pre-processed temperatures needed for object temperature calculation */
    pre_ambient = mlx90632_preprocess_temp_ambient_extended(ambient_new_raw,
                                            ambient_old_raw, Gb);
    pre_object = mlx90632_preprocess_temp_object_extended(object_new_raw,
                                           ambient_new_raw, ambient_old_raw,
                                           Ka);
    /* Calculate object temperature */
    object = mlx90632_calc_temp_object_extended(pre_object, pre_ambient, ambient, Ea, Eb, Ga, Fa, Fb, Ha,
Hb);

   /* Do something with the temperature data */

   /* The measurement is done and the MLX90632 device is in sleep mode
    * Waiting here will determine the active to sleep mode ratio, which would
    * determine the average power consumption over time */
}
```

6 Conclusion

The MLX90632 temperature sensor is a flexible device that can be used in a wide range of applications. The different measurement modes allow for a low power consumption while maintaining the high accuracy of the calculated temperatures.
Melexis provides an API that is publicly available at https://github.com/melexis/mlx90632-library.git

7 Revision history

Revision	Date	Change history
001	15-Feb-23	Creation

Table 12 – Revision history

8 Disclaimer

The content of this document is believed to be correct and accurate. However, the content of this document is furnished "as is" for informational use only and no representation, nor warranty is provided by Melexis about its accuracy, nor about the results of its implementation. Melexis assumes no responsibility or liability for any errors or inaccuracies that may appear in this document. Customer will follow the practices contained in this document under its sole responsibility. This documentation is in fact provided without warranty, term, or condition of any kind, either implied or expressed, including but not limited to warranties of merchantability, satisfactory quality, non-infringement, and fitness for purpose. Melexis, its employees and agents and its affiliates' and their employees and agents will not be responsible for any loss, however arising, from the use of, or reliance on this document. Notwithstanding the foregoing, contractual obligations expressly undertaken in writing by Melexis prevail over this disclaimer.

This document is subject to change without notice, and should not be construed as a commitment by Melexis. Therefore, before placing orders or prior to designing the product into a system, users or any third party should obtain the latest version of the relevant information.
Users or any third party must determine the suitability of the product described in this document for its application, including the level of reliability required and determine whether it is fit for a particular purpose.

This document as well as the product here described may be subject to export control regulations. Be aware that export might require a prior authorization from competent authorities. The product is not designed, authorized or warranted to be suitable in applications requiring extended temperature range and/or unusual environmental requirements. High reliability applications, such as medical life-support or life-sustaining equipment or avionics application are specifically excluded by Melexis. The product may not be used for the following applications subject to export control regulations: the development, production, processing, operation, maintenance, storage, recognition or proliferation of:
1. chemical, biological or nuclear weapons, or for the development, production, maintenance or storage of missiles for such weapons;
2. civil firearms, including spare parts or ammunition for such arms;
3. defense related products, or other material for military use or for law enforcement;
4. any applications that, alone or in combination with other goods, substances or organisms could cause serious harm to persons or goods and that can be used as a means of violence in an armed conflict or any similar violent situation.

No license nor any other right or interest is granted to any of Melexis' or third party's intellectual property rights.

If this document is marked “restricted” or with similar words, or if in any case the content of this document is to be reasonably understood as being confidential, the recipient of this document shall not communicate, nor disclose to any third party, any part of the document without Melexis’ express written consent. The recipient shall take all necessary measures to apply and preserve the confidential character of the document. In particular, the recipient shall (i) hold document in confidence with at least the same degree of care by which it maintains the confidentiality of its own proprietary and confidential information, but no less than reasonable care; (ii) restrict the disclosure of the document solely to its employees for the purpose for which this document was received, on a strictly need to know basis and providing that such persons to whom the document is disclosed are bound by confidentiality terms substantially similar to those in this disclaimer; (iii) use the document only in connection with the purpose for which this document was received, and reproduce document only to the extent necessary for such purposes; (iv) not use the document for commercial purposes or to the detriment of Melexis or its customers. The confidentiality obligations set forth in this disclaimer will have indefinite duration and in any case they will be effective for no less than 10 years from the receipt of this document.

This disclaimer will be governed by and construed in accordance with Belgian law and any disputes relating to this disclaimer will be subject to the exclusive jurisdiction of the courts of Brussels, Belgium.

The invalidity or ineffectiveness of any of the provisions of this disclaimer does not affect the validity or effectiveness of the other provisions.
The previous versions of this document are repealed.

IATF 16949 and ISO 14001 Certified

Happy to help you! www.melexis.com/technical-inquiry