

Specification document of LM50B

Component managed of the features	Component manuf	facturer '	Гexas	Instrumer
-----------------------------------	-----------------	------------	-------	-----------

Model number LM50B

Datasheets LM50 and LM50-Q1 SOT-23 Single-Supply Centigrade

Temperature Sensor datasheet (Rev. G)

Specification Ver 01.00.00 Nov 1,2022 New release Documentation provided Rui Long Lab Inc. https://rui-long-lab.com/

1.	Component datasheet	2
	·	
2.	Component Software IF specification	3
3	File Structure and Definitions	5

License

Open Source Software for Embedded Components ("OSS-EC") is open source software files and related documentation files for component products used in computer systems and other applications. OSS-EC is provided to those who accept the OSS-EC Terms of Use for the OSS-EC site; see https://oss-ec.com/license_agreement/ for the OSS-EC Terms of Use. By downloading the OSS-EC from the OSS-EC site or obtaining the OSS-EC by any means, you accept the Terms of Use. Please read and accept the Terms of Use before using the OSS-EC. If you do not agree to the Terms of Use, please do not use the OSS-EC.



1. Component datasheet

Temperature accuracy ± 2.0 ° C Accuracy $T_A = 25$ ° C

 ± 3.0 ° C Accuracy $T_A = T_{\text{MAX}}(100^{\circ} \text{ C})$

 -3.5° C $\sim 3^{\circ}$ C Accuracy T_A = T_{MIN}(-25° C)

Temperature range $-25 \text{ to } +100^{\circ} \text{ C}$

Range of power supply voltage (Vdd) 4.5 to 10.0[V]

Output voltage (Vout) Linear $10 \text{ [mV/}^{\circ} \text{ C] Typ. (-25 to } +100^{\circ} \text{ C)}$

0 [° C] 500 [mV]

Calculation $Vout = 0.5V + (0.01 \text{ V/}^{\circ} \text{ C} \times \text{Ta})$

 $Ta = (Vout - 0.5V) / (0.01 V/^{\circ} C)$

Vdd vs Vout Non-link

Applications IoT etc

• Computers

· Disk Drives

· Battery Management

FAX Machines

Printers

· Portable Medical Instruments

· HVAC

• Power Supply Modules



2. Component Software IF specification

The software interface specifications based on the LM50B component specifications are as follows. The voltage value-to-physical value conversion equation is a linear conversion equation as shown in the equation below.

ADC value to voltage value conversion formula

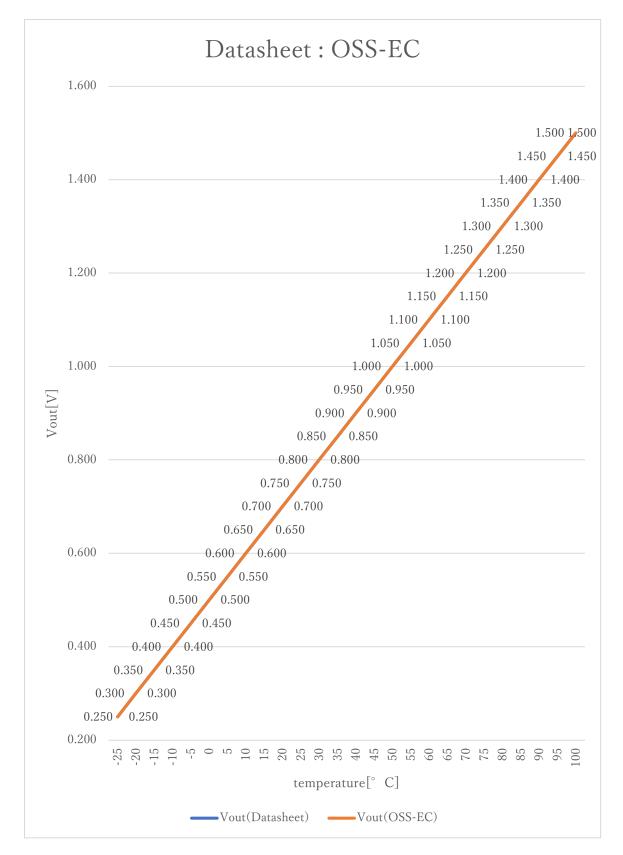
$$vi = (ai \times iADC_vdd) / 2^{iADC_bit}$$
 [V]

Voltage value to physical value conversion formula

y = (vi - iLM50B_xoff) / iLM50B_gain + iLM50B_yoff [°C] iLM50B_min
$$\leq$$
 y \leq iLM50B_max

```
A/D conversion value
ai
                  Sensor output voltage value [V]
٧i
i ADC_vdd
                  Sensor supply voltage value [V]
iADC bit
                  A/D conversion bit length
                  Temperature value [°C]
#define iLM50B_xoff
                                                      // X offset [V]
                                    0. 5F
                                                      // Y offset [°C]
#define iLM50B_yoff
                                    0. OF
#define iLM50B_gain
                                    <u>0. 01F</u>
                                                      // Gain [V/°C]
                                                      // Temperature Max [°C]
#define iLM50B_max
                                    100. OF
#define iLM50B_min
                                    <u>-25. 0F</u>
                                                      // Temperature Min [°C]
```





 $Vout(Datasheet) = 10 \text{ mV/}^{\circ} \text{ C} \times \text{ T}^{\circ} \text{ C} + 500 \text{ mV}$



3. File Structure and Definitions

LM50B.h

```
#include "user_define.h"
// Components number
#define iLM50B
                             130U
                                                           // Texas Instruments LM50B
// LM50B System Parts definitions
#define iLM50B_xoff
                             <u>0. 5F</u>
                                                           // X offset [V]
#define iLM50B_yoff
                             <u>0. 0F</u>
                                                           // Y offset [°C]
#define iLM50B_gain
                             0.01F
                                                           // Gain [V/°C]
#define iLM50B_max
                             100. OF
                                                           // Temperature Max [°C]
#define iLM50B_min
                             <u>-25. 0F</u>
                                                           // Temperature Min [°C]
extern const tbl_adc_t tbl_LM50B;
```



LM50B.cpp

```
#include
                "LM50B. h"
#if
        iLM50B_ma == iSMA
                                                         // Simple moving average filter
static float32 LM50B_sma_buf[iLM50B_SMA_num];
static const sma_f32_t LM50B_Phy_SMA =
        iInitial ,
                                                         // Initial state
        iLM50B_SMA_num ,
                                                       // Simple moving average number & buf size
        OU ,
                                                         // buffer position
        0.0F,
                                                         // sum
        &LM50B_sma_buf[0]
                                                         // buffer
};
#elif
        iLM50B_ma == iEMA
                                                         // Exponential moving average filter
static const ema_f32_t LM50B_Phy_EMA =
{
        iInitial ,
                                                         // Initial state
        0.0F,
                                                         // Xn-1
        iLM50B_EMA_K
                                                         // Exponential smoothing factor
};
#elif
        iLM50B_ma == iWMA
                                                         // Weighted moving average filter
static float32 LM50B_wma_buf[iLM50B_WMA_num];
static const wma_f32_t LM50B_Phy_WMA =
{
        iInitial ,
                                                         // Initial state
        iLM50B_WMA_num ,
                                               // Weighted moving average number & buf size
                                                         // buffer poition
        OU ,
        iLM50B_WMA_num * (iLM50B_WMA_num + 1)/2,
                                                         // kn sum
        &LM50B_wma_buf[0]
                                                         // Xn buffer
};
#else
                                                         // Non-moving average filter
#endif
#define iDummy_adr
                         0xffffffff
                                                         // Dummy address
```



```
const tbl_adc_t tbl_LM50B =
        iLM50B
        iLM50B_pin
        iLM50B\_xoff
        iLM50B_yoff
        iLM50B_gain
        iLM50B_max
        iLM50B_min
        iLM50B_ma
#if
        iLM50B_ma == iSMA
                                                          // Simple moving average filter
        &LM50B_Phy_SMA
        (ema_f32_t*) iDummy_adr,
        (wma_f32_t*) iDummy_adr
#elif
        iLM50B_ma == iEMA
                                                          // Exponential moving average filter
        (sma_f32_t*) iDummy_adr,
        &LM50B_Phy_EMA
        (wma_f32_t*) iDummy_adr
#elif
        iLM50B_ma == iWMA
                                                          // Weighted moving average filter
        (sma_f32_t*) iDummy_adr ,
        (ema_f32_t*) iDummy_adr ,
        &LM50B_Phy_WMA
#else
                                                          // Non-moving average filter
         (sma_f32_t*)iDummy_adr ,
         (ema_f32_t*) iDummy_adr ,
        (wma_f32_t*) iDummy_adr
#endif
};
```