

Component manufacturer

Specification document of LM50C, LM50-Q1

Model number	LM50C	, LM50	150-Q1				
Datasheets	LM50	and	LM50-O1	SOT-23	Single-Supply	Centign	

Texas Instruments

Temperature Sensor datasheet (Rev. G)

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

1.	Component datasheet	2
	Component Software IF specification	
۷.	Component Software in Specification	
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://ossec.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 $\pm 3.0^{\circ}$ C Accuracy $T_A = 25^{\circ}$ C

 $\pm 4.0 \, ^{\circ} \ C \qquad \qquad Accuracy \ T_A = T_{\text{\tiny MAX}}(125^{\circ} \ C)$

 ± 4.0 ° C Accuracy $T_A = T_{MIN}(-40$ ° C)

Temperature range $-40 \text{ to } +125^{\circ} \text{ C}$

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

Output voltage (Vout) Linear $10 \text{ [mV/}^{\circ} \text{ C] Typ. (-40 to +125}^{\circ} \text{ C)}$

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

Automotive



2. Component Software IF specification

The software interface specifications based on the LM50C, LM50-Q1 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

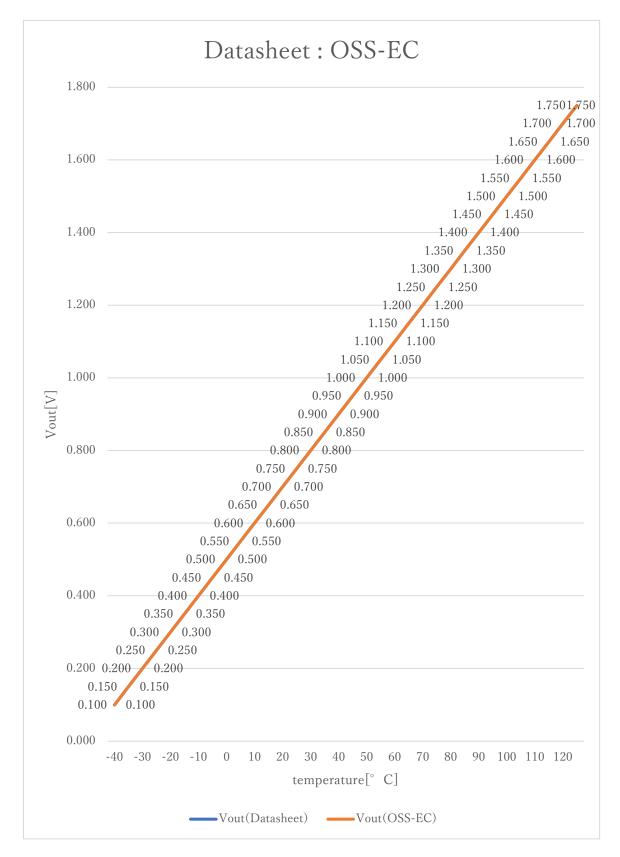
$$vi = (ai \times iADC_vdd) / 2^{iADC_bit}$$
 [V]

Voltage value to physical value conversion formula

$$y = (vi - iLM50C_xoff) / iLM50C_gain + iLM50C_yoff$$
 [°C] $iLM50C_min \le y \le iLM50C_max$

```
A/D conversion value
ai
٧i
                 Sensor output voltage value [V]
i ADC vdd
                 Sensor supply voltage value [V]
iADC_bit
                 A/D conversion bit length
                 Temperature value [°C]
#define iLM50C_xoff
                                                     // X offset [V]
                                   0. 5F
#define iLM50C_yoff
                                   <u>0. 0F</u>
                                                     // Y offset [°C]
                                                     // Gain [V/°C]
#define iLM50C_gain
                                   0. 01F
#define iLM50C_max
                                   125. OF
                                                     // Temperature Max [°C]
#define iLM50C_min
                                   -40. 0F
                                                     // 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

LM50C.h

```
#include "user_define.h"
// Components number
#define iLM50C
                             131U
                                                           // Texas Instruments LM50C, LM50-Q1
// LM50C, LM50-Q1 System Parts definitions
#define iLM50C_xoff
                             <u>0. 5F</u>
                                                           // X offset [V]
#define iLM50C_yoff
                             <u>0. 0F</u>
                                                           // Y offset [°C]
#define iLM50C_gain
                             0.01F
                                                           // Gain [V/°C]
#define iLM50C_max
                             125. OF
                                                           // Temperature Max [°C]
#define iLM50C_min
                             <u>-40. 0F</u>
                                                           // Temperature Min [°C]
extern const tbl_adc_t tbl_LM50C;
```



LM50C.cpp

```
#include
                "LM50C. h"
#if
        iLM50C_ma == iSMA
                                                        // Simple moving average filter
static float32 LM50C_sma_buf[iLM50C_SMA_num];
static const sma_f32_t LM50C_Phy_SMA =
        iInitial ,
                                                        // Initial state
        iLM50C_SMA_num ,
                                                       // Simple moving average number & buf size
        OU ,
                                                         // buffer position
        0.0F,
                                                        // sum
        &LM50C_sma_buf[0]
                                                        // buffer
};
#elif
        iLM50C_ma == iEMA
                                                         // Exponential moving average filter
static const ema_f32_t LM50C_Phy_EMA =
{
        iInitial ,
                                                        // Initial state
        0.0F,
                                                         // Xn-1
        iLM50C_EMA_K
                                                        // Exponential smoothing factor
};
#elif
        iLM50C_ma == iWMA
                                                        // Weighted moving average filter
static float32 LM50C_wma_buf[iLM50C_WMA_num];
static const wma_f32_t LM50C_Phy_WMA =
{
        iInitial ,
                                                         // Initial state
        iLM50C_WMA_num ,
                                               // Weighted moving average number & buf size
                                                        // buffer poition
        OU ,
                                                        // kn sum
        iLM50C_WMA_num * (iLM50C_WMA_num + 1)/2,
        &LM50C_wma_buf[0]
                                                        // Xn buffer
};
#else
                                                         // Non-moving average filter
#endif
#define iDummy_adr
                         0xffffffff
                                                        // Dummy address
```



```
const tbl_adc_t tbl_LM50C =  \frac{1}{2} 
        iLM50C
        iLM50C_pin
        iLM50C\_xoff
        iLM50C\_yoff
        iLM50C_gain
        iLM50C_max
        iLM50C_min
        iLM50C_ma
#if
        iLM50C_ma == iSMA
                                                           // Simple moving average filter
        &LM50C_Phy_SMA
         (ema_f32_t*) iDummy_adr ,
         (wma_f32_t*) iDummy_adr
#elif
                                                           // Exponential moving average filter
        iLM50C_ma == iEMA
         (sma_f32_t*) iDummy_adr,
        &LM50C_Phy_EMA
         (wma_f32_t*) iDummy_adr
#elif
        iLM50C_ma == iWMA
                                                            // Weighted moving average filter
         (sma_f32_t*) iDummy_adr ,
         (ema_f32_t*) iDummy_adr ,
        &LM50C_Phy_WMA
#else
                                                            // Non-moving average filter
         (sma_f32_t*) iDummy_adr ,
         (ema_f32_t*) iDummy_adr ,
         (wma_f32_t*) iDummy_adr
#endif
};
```