

Specification document of LM35D

Component manufacturer Texas instrument	Component man	nufacturer	Texas	Instrument
---	---------------	------------	-------	------------

Model number LM35D

Datasheets LM35 Precision Centigrade Temperature Sensors datasheet (Rev.

H)

Specification Ver 01.00.00 Nov 3,2022 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 ± 0.6 ° C Typ Accuracy $T_A = 25$ ° C

 $\pm\,0.9\,^{\circ}\ C\quad Typ\qquad \qquad Accuracy\ T_A = T_{\text{\tiny MAX}}(100^{\circ}\ C)$

 $\pm 0.9\,^{\circ}$ C Typ Accuracy $T_A = T_{\text{MIN}}(0^{\circ}$ C)

Temperature range $0 \text{ to } +100^{\circ} \text{ C}$

Range of power supply voltage (Vdd) 4.0 to 30.0[V]

Output voltage (Vout) Linear 10 [mV/° C] Typ

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

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

Vdd vs Vout Non-link

Applications IoT etc

· Power Supplies

· Battery Management

• HVAC

Aplicances



2. Component Software IF specification

The software interface specifications based on the LM35D 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]

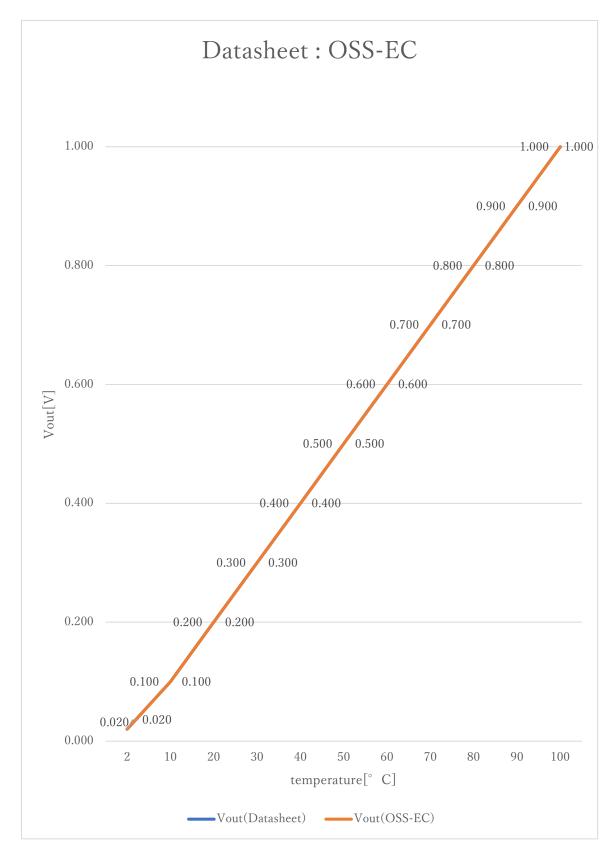
Voltage value to physical value conversion formula

$$y = (vi - iLM35D_xoff) / iLM35D_gain + iLM35D_yoff [°C]$$

$$iLM35D_min \le y \le iLM35D_max$$

```
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 iLM35D_xoff
                                                     // X offset [V]
                                   0. 0F
                                                     // Y offset [°C]
#define iLM35D_yoff
                                   0. OF
#define iLM35D_gain
                                   <u>0.01F</u>
                                                     // Gain [V/°C]
                                                     // Temperature Max [°C]
#define iLM35D_max
                                   100. OF
#define iLM35D_min
                                   2. 0F
                                                     // Temperature Min [°C]
```





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



3. File Structure and Definitions

LM35D.h

```
#include "user_define.h"
// Components number
#define iLM35D
                                                           // Texas Instruments LM35D
                             128U
// LM35D System Parts definitions
#define iLM35D_xoff
                             <u>0. 0F</u>
                                                          // X offset [V]
#define iLM35D_yoff
                             <u>0. 0F</u>
                                                          // Y offset [°C]
#define iLM35D_gain
                             0.01F
                                                          // Gain [V/°C]
#define iLM35D_max
                             100. 0F
                                                          // Temperature Max [°C]
#define iLM35D_min
                             2. OF
                                                          // Temperature Min [°C]
extern const tbl_adc_t tbl_LM35D;
```



LM35D.cpp

```
#include
                "LM35D. h"
#if
        iLM35D_ma == iSMA
                                                        // Simple moving average filter
static float32 LM35D_sma_buf[iLM35D_SMA_num];
static const sma_f32_t LM35D_Phy_SMA =
        iInitial ,
                                                        // Initial state
        iLM35D_SMA_num ,
                                                       // Simple moving average number & buf size
        OU ,
                                                         // buffer position
        0.0F,
                                                        // sum
        &LM35D_sma_buf[0]
                                                        // buffer
};
#elif
        iLM35D_ma == iEMA
                                                         // Exponential moving average filter
static const ema_f32_t LM35D_Phy_EMA =
{
        iInitial ,
                                                        // Initial state
        0.0F,
                                                         // Xn-1
        iLM35D_EMA_K
                                                        // Exponential smoothing factor
};
#elif
        iLM35D_ma == iWMA
                                                        // Weighted moving average filter
static float32 LM35D_wma_buf[iLM35D_WMA_num];
static const wma_f32_t LM35D_Phy_WMA =
{
        iInitial ,
                                                         // Initial state
        iLM35D_WMA_num ,
                                               // Weighted moving average number & buf size
                                                        // buffer poition
        OU ,
        iLM35D_WMA_num * (iLM35D_WMA_num + 1)/2,
                                                        // kn sum
        &LM35D_wma_buf[0]
                                                        // Xn buffer
};
#else
                                                         // Non-moving average filter
#endif
#define iDummy_adr
                         0xffffffff
                                                        // Dummy address
```



```
const tbl_adc_t tbl_LM35D =
        iLM35D
        iLM35D_pin
        iLM35D\_xoff
        iLM35D_yoff
        iLM35D_gain
        iLM35D_max
        iLM35D_min
        iLM35D_ma
#if
        iLM35D_ma == iSMA
                                                          // Simple moving average filter
        &LM35D_Phy_SMA
        (ema_f32_t*) iDummy_adr,
        (wma_f32_t*) iDummy_adr
#elif
        iLM35D_ma == iEMA
                                                          // Exponential moving average filter
        (sma_f32_t*) iDummy_adr,
        &LM35D_Phy_EMA
        (wma_f32_t*) iDummy_adr
#elif
        iLM35D_ma == iWMA
                                                          // Weighted moving average filter
        (sma_f32_t*) iDummy_adr ,
        (ema_f32_t*) iDummy_adr ,
        &LM35D_Phy_WMA
#else
                                                          // Non-moving average filter
         (sma_f32_t*)iDummy_adr ,
         (ema_f32_t*) iDummy_adr ,
        (wma_f32_t*) iDummy_adr
#endif
};
```