Specification document of LM35C, LM35CA

Component manufacturer	Texas Instruments
Model number	LM35C, LM35CA
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/

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	±0.4° C Typ	Accuracy T_A = 25° C
	±0.8° C Typ	Accuracy T_A = T_{MAX}(110° C)
	±0.8° C Typ	Accuracy T_A = T_{MIN}(-40° C)
Temperature range	-40 to +110° 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 V/° C × Ta	
	Ta = Vout / (0.01 V/° 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 LM35C, LM35CA 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

$$vi = (ai \times iADC\_vdd) / 2^{iADC\_bit} \quad [V]$$

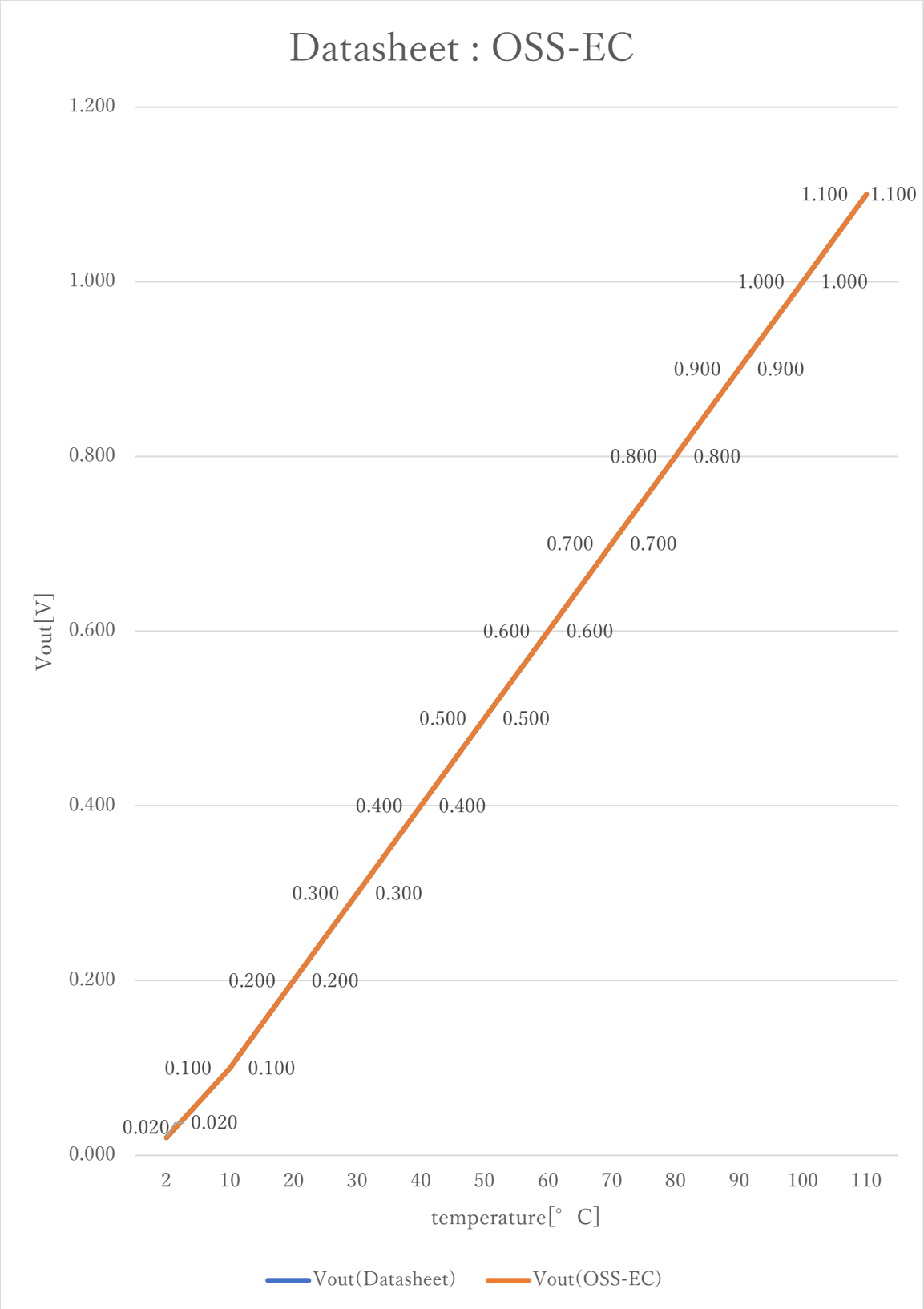
Voltage value to physical value conversion formula

$$y = (vi - iLM35C\_xoff) / iLM35C\_gain + iLM35C\_yoff \quad [℃]$$

$$iLM35C\_min \leqq y \leqq iLM35C\_max$$

ai	A/D conversion value
vi	Sensor output voltage value [V]
iADC_vdd	Sensor supply voltage value [V]
iADC_bit	A/D conversion bit length
y	Temperature value [℃]

```
#define iLM35C_xoff             0.0F          // X offset [V]
#define iLM35C_yoff             0.0F          // Y offset [℃]
#define iLM35C_gain             0.01F         // Gain [V/℃]
#define iLM35C_max              110.0F        // Temperature Max [℃]
#define iLM35C_min              2.0F          // Temperature Min [℃]
                                              // CAUTION:-40[° C],the circuit
                                                  needs a voltage Offset
```



Vout(Datasheet) = 10 mV/° C × T ° C

3. File Structure and Definitions

LM35C.h

```
#include "user_define.h"

// Components number
#define iLM35C              127U                    // Texas Instruments LM35C, LM35CA

// LM35C、LM35CA System Parts definitions
#define iLM35C_xoff         0.0F                    // X offset [V]
#define iLM35C_yoff         0.0F                    // Y offset [℃]
#define iLM35C_gain         0.01F                   // Gain [V/℃]
#define iLM35C_max          110.0F                  // Temperature Max [℃]
#define iLM35C_min          2.0F                    // Temperature Min [℃]
                                                    // CAUTION:-40[° C],the circuit
                                                       needs a voltage Offset


extern const tbl_adc_t tbl_LM35C;
```

LM35C.cpp

```
#include        "LM35C.h"
#if     iLM35C_ma == iSMA                               // Simple moving average filter
static float32 LM35C_sma_buf[iLM35C_SMA_num];
static const sma_f32_t LM35C_Phy_SMA =
{
        iInitial ,                                      // Initial state
        iLM35C_SMA_num ,                                // Simple moving average number & buf size
        0U ,                                            // buffer position
        0.0F ,                                          // sum
        &LM35C_sma_buf[0]                               // buffer
};
#elif   iLM35C_ma == iEMA                               // Exponential moving average filter
static const ema_f32_t LM35C_Phy_EMA =
{
        iInitial ,                                      // Initial state
        0.0F ,                                          // Xn-1
        iLM35C_EMA_K                                    // Exponential smoothing factor
};
#elif   iLM35C_ma == iWMA                               // Weighted moving average filter
static float32 LM35C_wma_buf[iLM35C_WMA_num];
static const wma_f32_t LM35C_Phy_WMA =
{
        iInitial ,                                      // Initial state
        iLM35C_WMA_num ,                        // Weighted moving average number & buf size
        0U ,                                            // buffer poition
        iLM35C_WMA_num * (iLM35C_WMA_num + 1)/2 ,       // kn sum
        &LM35C_wma_buf[0]                               // Xn buffer
};
#else                                                   // Non-moving average filter
#endif

#define iDummy_adr      0xffffffff                      // Dummy address
```

```
const tbl_adc_t tbl_LM35C =
{
        iLM35C                  ,
        iLM35C_pin              ,
        iLM35C_xoff             ,
        iLM35C_yoff             ,
        iLM35C_gain             ,
        iLM35C_max              ,
        iLM35C_min              ,
        iLM35C_ma               ,

#if     iLM35C_ma == iSMA                       // Simple moving average filter
        &LM35C_Phy_SMA          ,
        (ema_f32_t*)iDummy_adr  ,
        (wma_f32_t*)iDummy_adr
#elif   iLM35C_ma == iEMA                       // Exponential moving average filter
        (sma_f32_t*)iDummy_adr  ,
        &LM35C_Phy_EMA          ,
        (wma_f32_t*)iDummy_adr
#elif   iLM35C_ma == iWMA                       // Weighted moving average filter
        (sma_f32_t*)iDummy_adr  ,
        (ema_f32_t*)iDummy_adr  ,
        &LM35C_Phy_WMA
#else                                           // Non-moving average filter
        (sma_f32_t*)iDummy_adr  ,
        (ema_f32_t*)iDummy_adr  ,
        (wma_f32_t*)iDummy_adr
#endif

};
```