Specification document of S-58LM20A

Component manufacturer	ABLIC		
Model number	S-58LM20A		
Datasheets	S-58LM20A Series TEMPERATURE SENSOR IC (ablic.com)		
Specification Ver	01.00.00	Sep 12,2022	New release
	01.01.00	Sep 29,2022	Component datasheet add Data correction
	01.01.01	Oct 18,2022	Corrected license content Application item add
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

Accuracy against temperature	±2.5° C (-55 ° C to +130° C)
Range of power supply voltage(Vdd)	2.4 to 5.5[V]
Output voltage (Vout)	Linear -11.77 [mV/° C] Typ. (-30° C to 130° C)
	-30[° C] 2.205 [V] Typ.
	30[° C] 1.515 [V] Typ.
	130[° C] 0.303 [V] Typ.
Vdd vs Vout	Non-link

Applications IoT etc

- Compensation of high-frequency circuits such as cellular phones and radio equipment
- Compensation of oscillation frequency in crystal oscillator
- LCD contrast compensation
- Compensation of amplifier gain
- Compensation of auto focus circuits
- Temperature detection in battery management
- Overheating prevention for charged batteries or halogen lights

2. Component Software IF specification

The software interface specifications based on the S-58LM20A 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

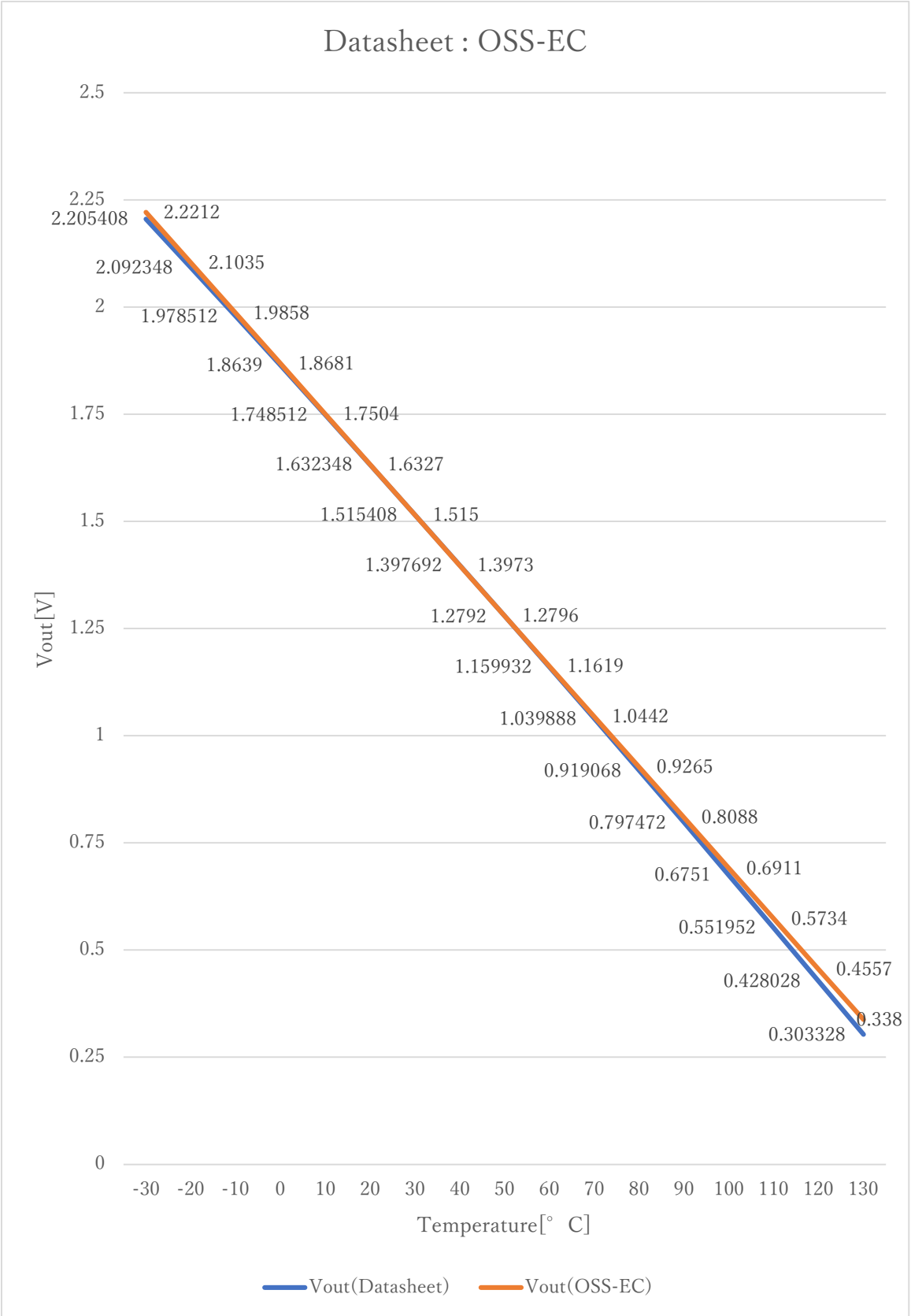
$$y = (vi - iS58LM20A\_xoff) / iS58LM20A\_gain + iS58LM20A\_yoff \quad [^\circ C]$$

$$iS58LM20A\_min \leqq y \leqq iS58LM20A\_max$$

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 [° C]

```
#define iS58LM20A_xoff      1.515F          // X offset [V]
#define iS58LM20A_yoff      30.0F           // Y offset [° C]
#define iS58LM20A_gain      -0.01177F       // Gain [V/° C]
#define iS58LM20A_max       130.0F          // Temperature Max [° C]
#define iS58LM20A_min       -30.0F          // Temperature Min [° C]
```

Note : Non-Linear iS58LM20A_min -55.0F



$$Vout(\ Datasheet\ ) = (\ -3.88 \times 10^{-6} \times T^2\ ) + (\ -1.15 \times 10^{-2} \times T\ ) + 1.8639\ V$$

3. File Structure and Definitions

S58LM20A.h

```
#include "user_define.h"

// Components number
#define iS58LM20A              103U                    // ABLIC S-58LM20A

// S58LM20A System Parts definitions
#define iS58LM20A_xoff         1.515F                  // X offset [V]
#define iS58LM20A_yoff         30.0F                   // Y offset [° C]
#define iS58LM20A_gain         -0.01177F               // Gain [V/° C]
#define iS58LM20A_max          130.0F                  // Temperature Max [° C]
#define iS58LM20A_min          -30.0F                  // Temperature Min [° C]

extern const tbl_adc_t tbl_S58LM20A;
```

S58LM20A.cpp

```
#include        "S58LM20A.h"
#if     iS58LM20A_ma == iSMA                            // Simple moving average filter
static float32 S58LM20A_sma_buf[iS58LM20A_SMA_num];
static const sma_f32_t S58LM20A_Phy_SMA =
{
        iInitial ,                                      // Initial state
        iS58LM20A_SMA_num ,                             // Simple moving average number & buf size
        0U ,                                            // buffer position
        0.0F ,                                          // sum
        &S58LM20A_sma_buf[0]                            // buffer
};
#elif   iS58LM20A_ma == iEMA                            // Exponential moving average filter
static const ema_f32_t S58LM20A_Phy_EMA =
{
        iInitial ,                                      // Initial state
        0.0F ,                                          // Xn-1
        iS58LM20A_EMA_K                                 // Exponential smoothing factor
};
#elif   iS58LM20A_ma == iWMA                            // Weighted moving average filter
static float32 S58LM20A_wma_buf[iS58LM20A_WMA_num];
static const wma_f32_t S58LM20A_Phy_WMA =
{
        iInitial ,                                      // Initial state
        iS58LM20A_WMA_num ,                             // Weighted moving average number & buf size
        0U ,                                            // buffer poition
        iS58LM20A_WMA_num * (iS58LM20A_WMA_num + 1)/2 , // kn sum
        &S58LM20A_wma_buf[0]                            // Xn buffer
};
#else                                                   // Non-moving average filter
#endif

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

```
const tbl_adc_t tbl_S58LM20A =
{
        iS58LM20A               ,
        iS58LM20A_pin           ,
        iS58LM20A_xoff          ,
        iS58LM20A_yoff          ,
        iS58LM20A_gain          ,
        iS58LM20A_max           ,
        iS58LM20A_min           ,
        iS58LM20A_ma            ,

#if     iS58LM20A_ma == iSMA                    // Simple moving average filter
        &S58LM20A_Phy_SMA       ,
        (ema_f32_t*)iDummy_adr  ,
        (wma_f32_t*)iDummy_adr
#elif   iS58LM20A_ma == iEMA                    // Exponential moving average filter
        (sma_f32_t*)iDummy_adr  ,
        &S58LM20A_Phy_EMA       ,
        (wma_f32_t*)iDummy_adr
#elif   iS58LM20A_ma == iWMA                    // Weighted moving average filter
        (sma_f32_t*)iDummy_adr  ,
        (ema_f32_t*)iDummy_adr  ,
        &S58LM20A_Phy_WMA
#else                                           // Non-moving average filter
        (sma_f32_t*)iDummy_adr  ,
        (ema_f32_t*)iDummy_adr  ,
        (wma_f32_t*)iDummy_adr
#endif

};
```