Specification document of S-5813A, S-5814A

Component manufacturer	ABLIC		
Model number	S-5813A, S-5814A		
Datasheets	S-5813A/5814A Series TEMPERATURE SENSOR IC (ablic.com)		
Specification Ver	01.00.00	Sep 30,2022	New release
	01.00.01	Oct 18,2022	Corrected license content Application item add
Documentation provided	Rui Long Lab Inc. https://rui-long-lab.com/		

1. Component datasheet

Accuracy against temperature	S-5813A ±5.0° C (-30 ° C to +100° C) S-5814A ±2.5° C (-30 ° C to +100° C)
Range of power supply voltage(Vdd)	2.9 to 10.0[V]
Output voltage (Vout)	Linear -11.04 [mV/° C] Typ. (-30° C to 100° C) Vdd = 5.0 [V] -30 [° C] 2.582 [V] Typ. 30 [° C] 1.940 [V] Typ. 100 [° C] 1.145 [V] Typ.
Vdd vs Vout	Non-link (ΔVout 0.006 to 0.007 [V])

Ta[° C]	Vdd[V]	Vout[V]
-40	3.00	2.677
	10.00	2.683
30	2.48	1.934
	10.00	1.940
100	2.48	1.142
	10.00	1.149

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-5813A/S-5814A 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

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

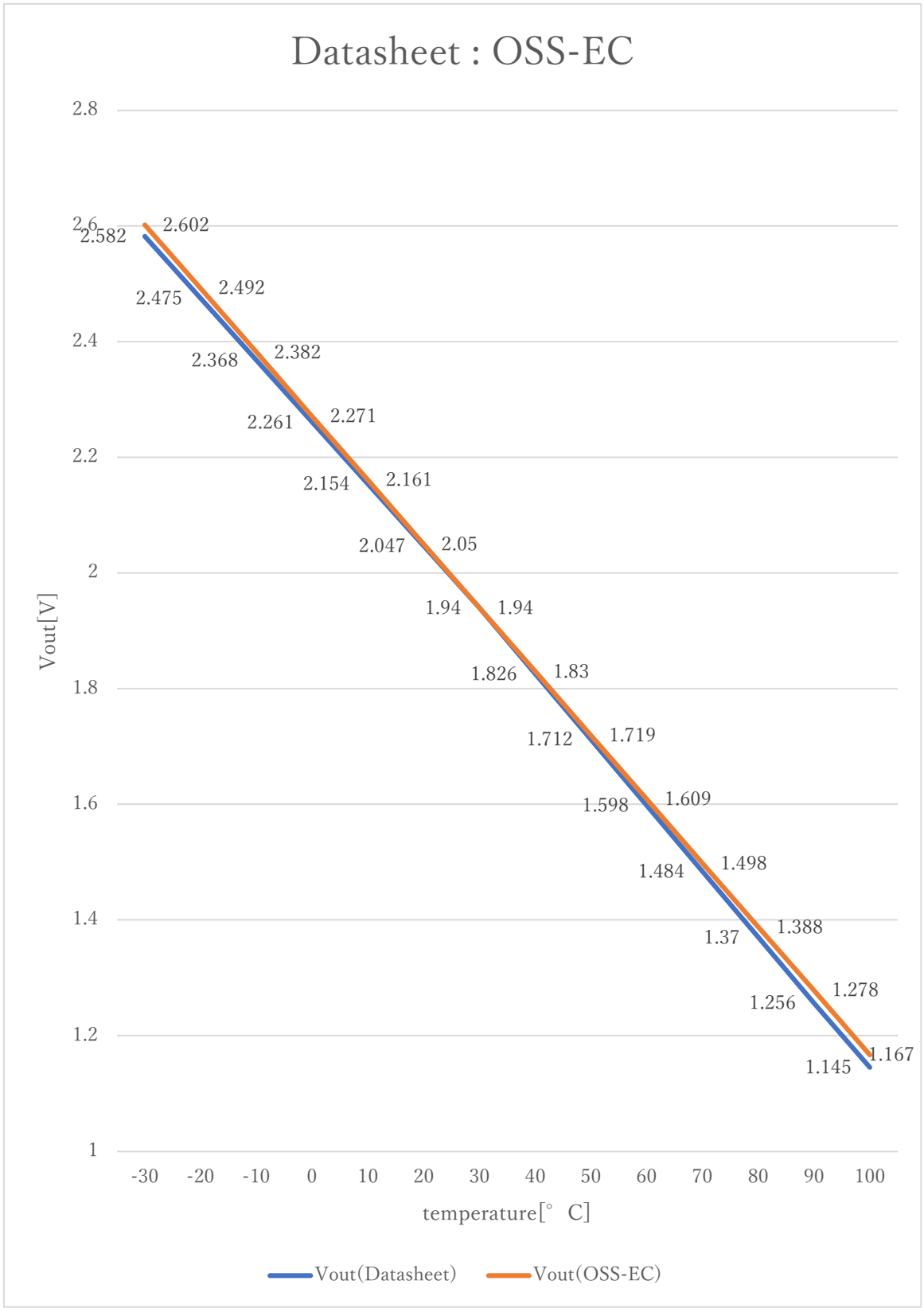
Voltage value to physical value conversion formula

$$y = (vi - iS5813A\_xoff) / iS5813A\_gain + iS5813A\_yoff \quad [℃]$$

$$iS5813A\_min \leqq y \leqq iS5813A\_max$$

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 iS5813A_xoff        1.940F          // X offset [V]
#define iS5813A_yoff        30.0F           // Y offset [℃]
#define iS5813A_gain        -0.01104F       // Gain [V/℃]
#define iS5813A_max         100.0F          // Temperature Max [℃]
#define iS5813A_min         -30.0F          // Temperature Min [℃]
```


Datasheet : OSS-EC
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
Vout[V]
2.602
2.582
2.492
2.475
2.382
2.368
2.271
2.261
2.161
2.154
2.05
2.047
1.94
1.94
1.83
1.826
1.719
1.712
1.609
1.598
1.498
1.484
1.388
1.37
1.278
1.256
1.167
1.145
-30 -20 -10 0 10 20 30 40 50 60 70 80 90 100
temperature[° C]
Vout(Datasheet)
Vout(OSS-EC)

3. File Structure and Definitions

S5813A.h

```
#include "user_define.h"

// Components number
#define iS5813A             104U                    // ABLIC S-5813A, S-5814A

// S-5813A,S-5814A System Parts definitions
#define iS5813A_xoff        1.940F                  // X offset [V]
#define iS5813A_yoff        30.0F                   // Y offset [℃]
#define iS5813A_gain        -0.01104F               // Gain [V/℃]
#define iS5813A_max         100.0F                  // Temperature Max [℃]
#define iS5813A_min         -30.0F                  // Temperature Min [℃]

extern const tbl_adc_t tbl_S5813A;
```

S5813A.cpp

```
#include        "S5813A.h"
#if     iS5813A_ma == iSMA                              // Simple moving average filter
static float32 S5813A_sma_buf[iS5813A_SMA_num];
static const sma_f32_t S5813A_Phy_SMA =
{
        iInitial ,                                      // Initial state
        iS5813A_SMA_num ,                               // Simple moving average number & buf size
        0U ,                                            // buffer position
        0.0F ,                                          // sum
        &S5813A_sma_buf[0]                              // buffer
};
#elif   iS5813A_ma == iEMA                              // Exponential moving average filter
static const ema_f32_t S5813A_Phy_EMA =
{
        iInitial ,                                      // Initial state
        0.0F ,                                          // Xn-1
        iS5813A_EMA_K                                   // Exponential smoothing factor
};
#elif   iS5813A_ma == iWMA                              // Weighted moving average filter
static float32 S5813A_wma_buf[iS5813A_WMA_num];
static const wma_f32_t S5813A_Phy_WMA =
{
        iInitial ,                                      // Initial state
        iS5813A_WMA_num ,                               // Weighted moving average number & buf size
        0U ,                                            // buffer poition
        iS5813A_WMA_num * (iS5813A_WMA_num + 1)/2 ,     // kn sum
        &S5813A_wma_buf[0]                              // Xn buffer
};
#else                                                   // Non-moving average filter
#endif

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

```
const tbl_adc_t tbl_S5813A =
{
        iS5813A             ,
        iS5813A_pin         ,
        iS5813A_xoff        ,
        iS5813A_yoff        ,
        iS5813A_gain        ,
        iS5813A_max         ,
        iS5813A_min         ,
        iS5813A_ma          ,

#if     iS5813A_ma == iSMA                      // Simple moving average filter
        &S5813A_Phy_SMA     ,
        (ema_f32_t*)iDummy_adr  ,
        (wma_f32_t*)iDummy_adr
#elif   iS5813A_ma == iEMA                      // Exponential moving average filter
        (sma_f32_t*)iDummy_adr  ,
        &S5813A_Phy_EMA     ,
        (wma_f32_t*)iDummy_adr
#elif   iS5813A_ma == iWMA                      // Weighted moving average filter
        (sma_f32_t*)iDummy_adr  ,
        (ema_f32_t*)iDummy_adr  ,
        &S5813A_Phy_WMA
#else                                             // Non-moving average filter
        (sma_f32_t*)iDummy_adr  ,
        (ema_f32_t*)iDummy_adr  ,
        (wma_f32_t*)iDummy_adr
#endif

};
```