Specification document of AD22103K

Component manufacturer	Analog Devices
Model number	AD22103K
Datasheets	AD22103 (Rev. B) (analog.com)
Specification Ver	01.00.00 Oct 04,2022 New release
	01.00.01 Oct 18,2022 Corrected license content Add Application item
Documentation provided	Rui Long Lab Inc. https://rui-long-lab.com/

1. Component datasheet

Temperature accuracy	±0.75° C (0° C to +100° C)
Range of power supply voltage (Vdd)	2.7 to 3.6[V]
Output voltage (Vout)	Linear 28.0×Vdd/3.3 [mV/° C] Typ.
	Vdd = 3.3 [V]
	0 [° C] 0.250[V] Typ.
	25 [° C] 0.950[V] Typ.
	100 [° C] 3.050 [V] Typ.
Calculation	Vout = (Vdd/3.3 V) × (0.25 V + 28.0 mV/° C × Ta)
	Ta = (Vout / (Vdd/3.3V)) – 0.25 V) / 28.0 mV/° C

Applications IoT etc

- Microprocessor thermal Management
- Battery and Low Powered Systems
- Power Supply Temperature Monitoring
- System Temperature Compensation
- Board Level Temperature Sensing

2. Component Software IF specification

The software interface specifications based on the AD22103K 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

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

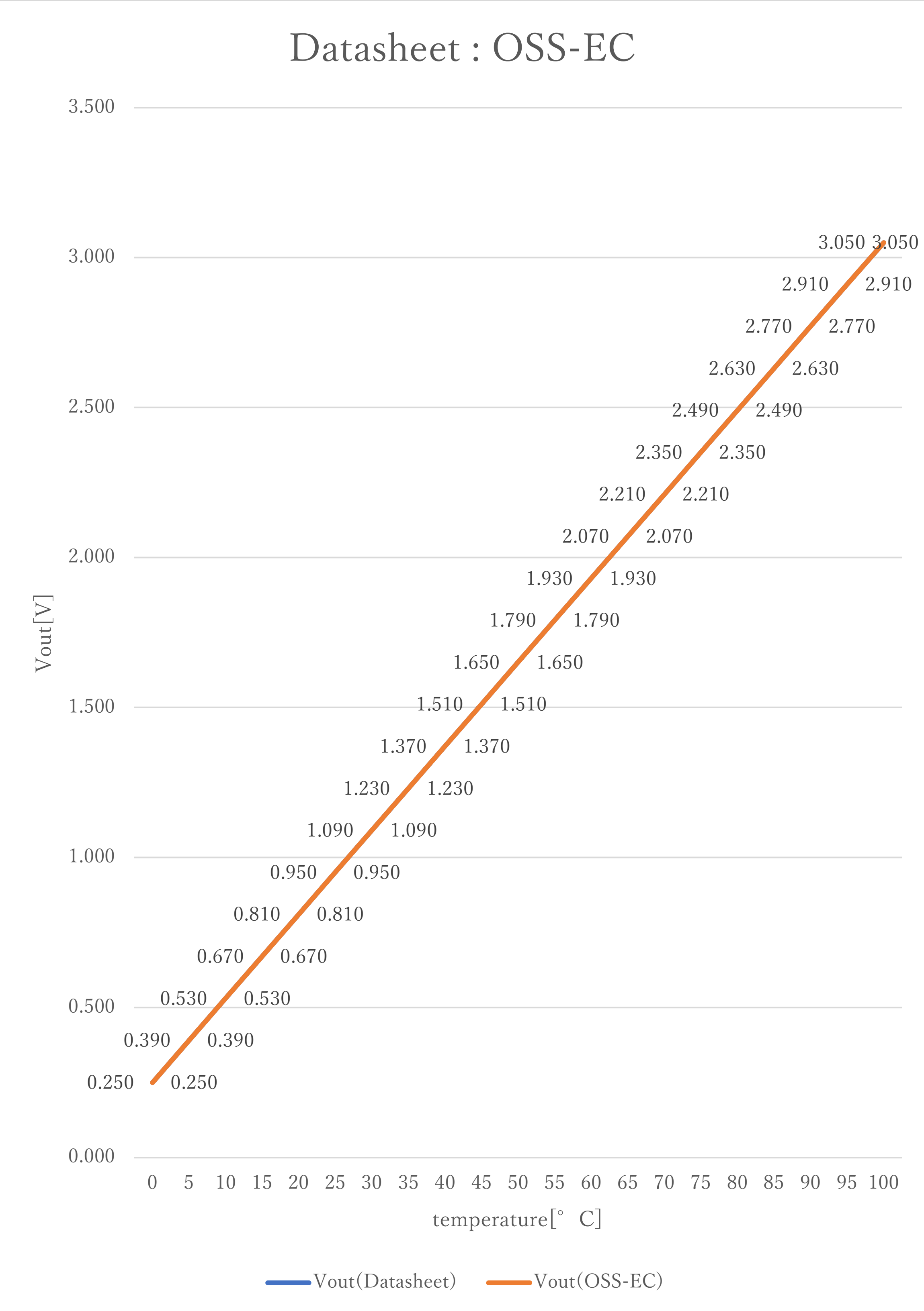
Voltage value to physical value conversion formula

$$y = (vi - iAD22103K\_xoff) / iAD22103K\_gain + iAD22103K\_yoff \quad [℃]$$

$$iAD22103K\_min \leqq y \leqq iAD22103K\_max$$

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 iAD22103K_xoff   (0.25F*(iADC_vdd/3.3))    // X offset [V]
#define iAD22103K_yoff   0.0F                      // Y offset [℃]
#define iAD22103K_gain   (0.028F*(iADC_vdd/3.3))   // Gain [V/℃]
#define iAD22103K_max    100.0F                    // Temperature Max [℃]
#define iAD22103K_min    0.0F                      // Temperature Min [℃]
```


Datasheet : OSS-EC
Vout[V]
3.500
3.000
2.500
2.000
1.500
1.000
0.500
0.000
0.250 0.250
0.390 0.390
0.530 0.530
0.670 0.670
0.810 0.810
0.950 0.950
1.090 1.090
1.230 1.230
1.370 1.370
1.510 1.510
1.650 1.650
1.790 1.790
1.930 1.930
2.070 2.070
2.210 2.210
2.350 2.350
2.490 2.490
2.630 2.630
2.770 2.770
2.910 2.910
3.050 3.050
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
temperature[° C]
Vout(Datasheet)
Vout(OSS-EC)

3. File Structure and Definitions

AD22103K.h

```
#include "user_define.h"

// Components number
#define iAD22103K            109U                    // Analog devices AD22103K

// AD22103K System Parts definitions
#define iAD22103K_xoff       (0.25F*(iADC_vdd/3.3))  // X offset [V]
#define iAD22103K_yoff       0.0F                    // Y offset [℃]
#define iAD22103K_gain       (0.028F*(iADC_vdd/3.3)) // Gain [V/℃]
#define iAD22103K_max        100.0F                  // Temperature Max [℃]
#define iAD22103K_min        0.0F                    // Temperature Min [℃]

extern const tbl_adc_t tbl_AD22103K;
```

AD22103K.cpp

```
#include        "AD22103K.h"
#if     iAD22103K_ma == iSMA                          // Simple moving average filter
static float32 AD22103K_sma_buf[iAD22103K_SMA_num];
static const sma_f32_t AD22103K_Phy_SMA =
{
        iInitial ,                                    // Initial state
        iAD22103K_SMA_num ,                          // Simple moving average number & buf size
        0U ,                                          // buffer position
        0.0F ,                                        // sum
        &AD22103K_sma_buf[0]                          // buffer
};
#elif   iAD22103K_ma == iEMA                          // Exponential moving average filter
static const ema_f32_t AD22103K_Phy_EMA =
{
        iInitial ,                                    // Initial state
        0.0F ,                                        // Xn-1
        iAD22103K_EMA_K                               // Exponential smoothing factor
};
#elif   iAD22103K_ma == iWMA                          // Weighted moving average filter
static float32 AD22103K_wma_buf[iAD22103K_WMA_num];
static const wma_f32_t AD22103K_Phy_WMA =
{
        iInitial ,                                    // Initial state
        iAD22103K_WMA_num ,                          // Weighted moving average number & buf size
        0U ,                                          // buffer poition
        iAD22103K_WMA_num * (iAD22103K_WMA_num + 1)/2 , // kn sum
        &AD22103K_wma_buf[0]                          // Xn buffer
};
#else                                                 // Non-moving average filter
#endif

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

```
const tbl_adc_t tbl_AD22103K =
{
        iAD22103K               ,
        iAD22103K_pin           ,
        iAD22103K_xoff          ,
        iAD22103K_yoff          ,
        iAD22103K_gain          ,
        iAD22103K_max           ,
        iAD22103K_min           ,
        iAD22103K_ma            ,

#if     iAD22103K_ma == iSMA                    // Simple moving average filter
        &AD22103K_Phy_SMA       ,
        (ema_f32_t*)iDummy_adr  ,
        (wma_f32_t*)iDummy_adr
#elif   iAD22103K_ma == iEMA                    // Exponential moving average filter
        (sma_f32_t*)iDummy_adr  ,
        &AD22103K_Phy_EMA       ,
        (wma_f32_t*)iDummy_adr
#elif   iAD22103K_ma == iWMA                    // Weighted moving average filter
        (sma_f32_t*)iDummy_adr  ,
        (ema_f32_t*)iDummy_adr  ,
        &AD22103K_Phy_WMA
#else                                           // Non-moving average filter
        (sma_f32_t*)iDummy_adr  ,
        (ema_f32_t*)iDummy_adr  ,
        (wma_f32_t*)iDummy_adr
#endif

};
```