Specification document of STLM20W87F

Component manufacturer	STMicroelectronics
Model number	STLM20W87F
Datasheets	Ultra-low current 2.4 V precision analog temperature sensor (st.com)
Specification Ver	01.00.00 Oct 20,2022 New release
Documentation provided	Rui Long Lab Inc. https://rui-long-lab.com/

1. Component datasheet

Temperature accuracy	±1.5° C maximum temperature accuracy at 25° C (±0.5° C typical)
Temperature range	-55 to +130° C
Range of power supply voltage (Vdd)	2.4 to 5.5[V]
Output voltage (Vout)	Linear -11.79 [mV/° C] (-55 to +130° C)

Calculation	Vout = 1.8528V + (-0.01179 V/° C × Ta)
	Ta = (Vout – 1.8528V) / (-0.01179 V/° C)

Vdd vs Vout	Non-link

Applications IoT etc

- Smartphones
- Multimedia PDA devices
- GPS devices
- Portable medical instruments
- Voltage-controlled crystal oscillator temperature monitors
- RF power transistor monitor

2. Component Software IF specification

The software interface specifications based on the STLM20W87F 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

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

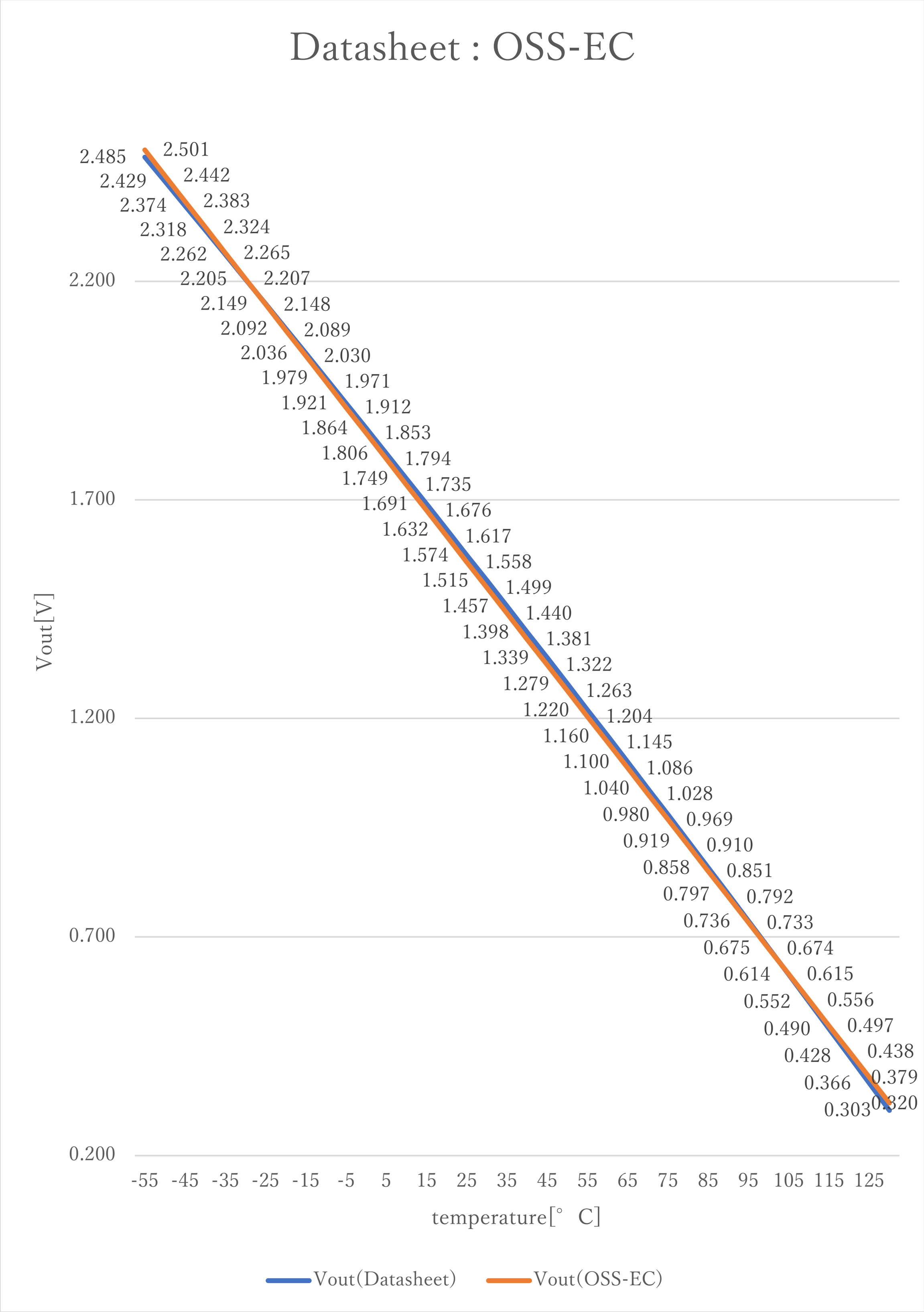
Voltage value to physical value conversion formula

$$\text{y} = (\text{vi} - \text{iSTLM20W87F\_xoff}) / \text{iSTLM20W87F\_gain} + \text{iSTLM20W87F\_yoff} \quad [{}^\circ\text{C}]$$

$$\text{iSTLM20W87F\_min} \leqq \text{y} \leqq \text{iSTLM20W87F\_max}$$

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 iSTLM20W87F_xoff        1.8528F         // X offset [V]
#define iSTLM20W87F_yoff        0.0F            // Y offset [℃]
#define iSTLM20W87F_gain        -0.01179F       // Gain [V/℃]
#define iSTLM20W87F_max         130.0F          // Temperature Max [℃]
#define iSTLM20W87F_min         -55.0F          // Temperature Min [℃]
```



Datasheet : OSS-EC

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

3. File Structure and Definitions

STLM20W87F.h

```
#include ″user_define.h″

// Components number
#define iSTLM20W87F         124U                      // STMicroelectronics STLM20W87F
// STLM20W87FSystem Parts definitions
#define iSTLM20W87F_xoff    1.8528F                   // X offset [V]
#define iSTLM20W87F_yoff    0.0F                      // Y offset [℃]
#define iSTLM20W87F_gain    -0.01179F                 // Gain [V/℃]
#define iSTLM20W87F_max     130.0F                    // Temperature Max [℃]
#define iSTLM20W87F_min     -55.0F                    // Temperature Min [℃]

extern const tbl_adc_t tbl_STLM20W87F;
```

STLM20DD9F.cpp

```
#include        ″STLM20W87F.h″
#if     iSTLM20W87F_ma == iSMA                          // Simple moving average filter
static float32 STLM20W87F_sma_buf[iSTLM20W87F_SMA_num];
static const sma_f32_t STLM20W87F_Phy_SMA =
{
        iInitial ,                                      // Initial state
        iSTLM20W87F_SMA_num ,                        // Simple moving average number & buf size
        0U ,                                            // buffer position
        0.0F ,                                          // sum
        &STLM20W87F_sma_buf[0]                          // buffer
};
#elif   iSTLM20W87F_ma == iEMA                          // Exponential moving average filter
static const ema_f32_t STLM20W87F_Phy_EMA =
{
        iInitial ,                                      // Initial state
        0.0F ,                                          // Xn-1
        iSTLM20W87F_EMA_K                               // Exponential smoothing factor
};
#elif   iSTLM20W87F_ma == iWMA                          // Weighted moving average filter
static float32 STLM20W87F_wma_buf[iSTLM20W87F_WMA_num];
static const wma_f32_t STLM20W87F_Phy_WMA =
{
        iInitial ,                                      // Initial state
        iSTLM20W87F_WMA_num ,                      // Weighted moving average number & buf size
        0U ,                                            // buffer poition
        iSTLM20W87F_WMA_num * (iSTLM20W87F_WMA_num + 1)/2 ,      // kn sum
        &STLM20W87F_wma_buf[0]                          // Xn buffer
};
#else                                                   // Non-moving average filter
#endif

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

```
const tbl_adc_t tbl_STLM20W87F =
{
        iSTLM20W87F            ,
        iSTLM20W87F_pin        ,
        iSTLM20W87F_xoff       ,
        iSTLM20W87F_yoff       ,
        iSTLM20W87F_gain       ,
        iSTLM20W87F_max        ,
        iSTLM20W87F_min        ,
        iSTLM20W87F_ma         ,

#if     iSTLM20W87F_ma == iSMA                          // Simple moving average filter
        &STLM20W87F_Phy_SMA    ,
        (ema_f32_t*)iDummy_adr ,
        (wma_f32_t*)iDummy_adr
#elif   iSTLM20W87F_ma == iEMA                          // Exponential moving average filter
        (sma_f32_t*)iDummy_adr ,
        &STLM20W87F_Phy_EMA    ,
        (wma_f32_t*)iDummy_adr
#elif   iSTLM20W87F_ma == iWMA                          // Weighted moving average filter
        (sma_f32_t*)iDummy_adr ,
        (ema_f32_t*)iDummy_adr ,
        &STLM20W87F_Phy_WMA
#else                                                   // Non-moving average filter
        (sma_f32_t*)iDummy_adr ,
        (ema_f32_t*)iDummy_adr ,
        (wma_f32_t*)iDummy_adr
#endif

};
```