Specification document of STLM20DD9F

Component manufacturer	STMicroelectronics
Model number	STLM20DD9F
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	-40 to +85°C
Range of power supply voltage (Vdd)	2.4 to 5.5[V]
Output voltage (Vout)	Linear -11.67 [mV/°C] (-40 to +85°C)
Calculation	Vout = 1.8583V + (-0.01167 V/°C × Ta)
	Ta = (Vout – 1.8583V) / (-0.01167 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 STLM20DD9F 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

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

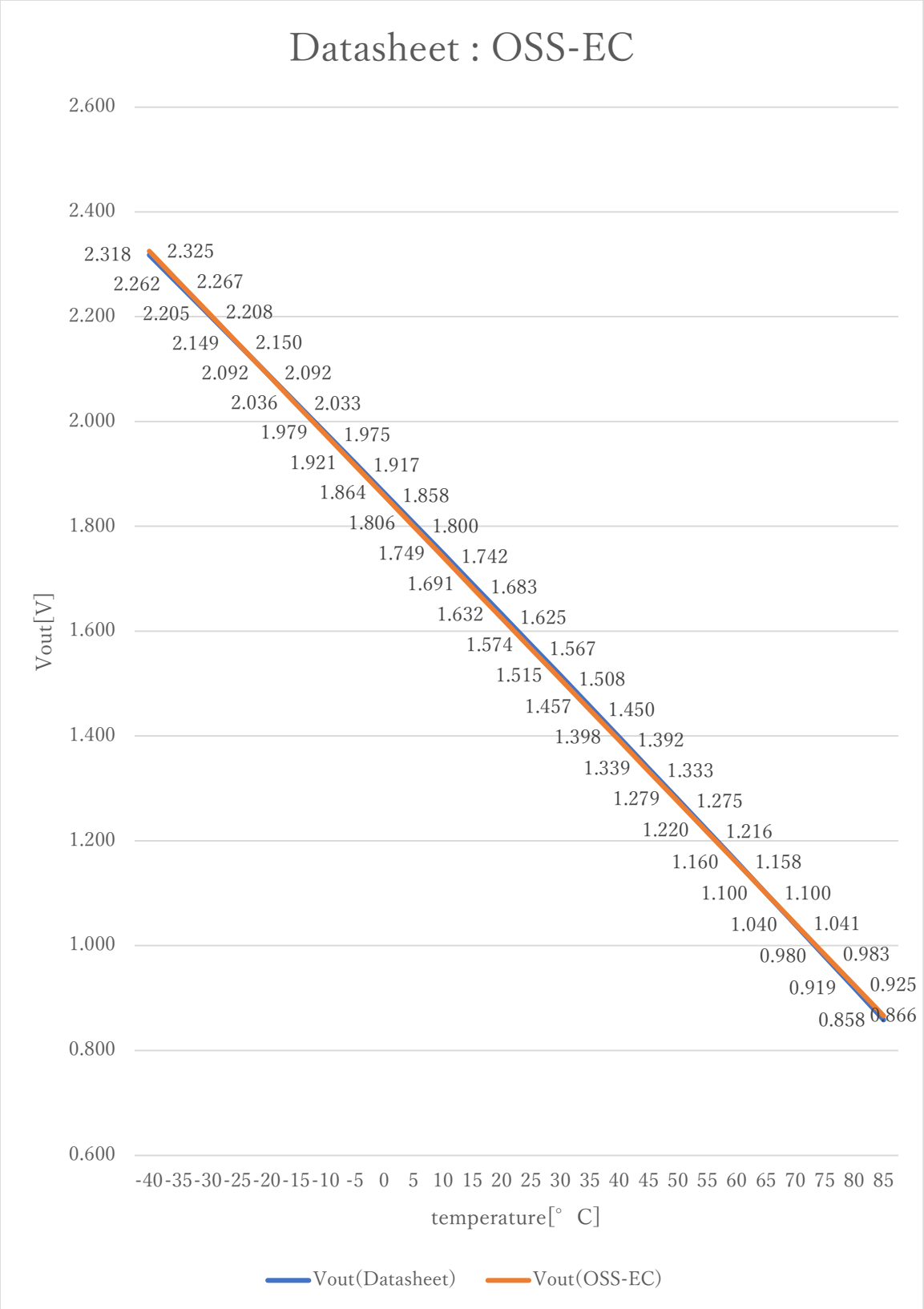
Voltage value to physical value conversion formula

$$y = (vi - iSTLM20DD9F\_xoff) / iSTLM20DD9F\_gain + iSTLM20DD9F\_yoff \quad [℃]$$

$$iSTLM20DD9F\_min \leqq y \leqq iSTLM20DD9F\_max$$

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 iSTLM20DD9F_xoff        1.8583F         // X offset [V]
#define iSTLM20DD9F_yoff        0.0F            // Y offset [℃]
#define iSTLM20DD9F_gain        -0.01167F       // Gain [V/℃]
#define iSTLM20DD9F_max         85.0F           // Temperature Max [℃]
#define iSTLM20DD9F_min         -40.0F          // Temperature Min [℃]
```



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

3. File Structure and Definitions

STLM20DD9F.h

```
#include "user_define.h"

// Components number
#define iSTLM20DD9F          123U                    // STMicroelectronics STLM20DD9F
// STLM20DD9F System Parts definitions
#define iSTLM20DD9F_xoff     1.8583F                 // X offset [V]
#define iSTLM20DD9F_yoff     0.0F                    // Y offset [℃]
#define iSTLM20DD9F_gain     -0.01167F               // Gain [V/℃]
#define iSTLM20DD9F_max      85.0F                   // Temperature Max [℃]
#define iSTLM20DD9F_min      -40.0F                  // Temperature Min [℃]

extern const tbl_adc_t tbl_STLM20DD9F;
```

STLM20DD9F.cpp

```
#include        "STLM20DD9F.h"
#if     iSTLM20DD9F_ma == iSMA                          // Simple moving average filter
static float32 STLM20DD9F_sma_buf[iSTLM20DD9F_SMA_num];
static const sma_f32_t STLM20DD9F_Phy_SMA =
{
        iInitial ,                                      // Initial state
        iSTLM20DD9F_SMA_num ,                          // Simple moving average number & buf size
        0U ,                                            // buffer position
        0.0F ,                                          // sum
        &STLM20DD9F_sma_buf[0]                          // buffer
};
#elif   iSTLM20DD9F_ma == iEMA                          // Exponential moving average filter
static const ema_f32_t STLM20DD9F_Phy_EMA =
{
        iInitial ,                                      // Initial state
        0.0F ,                                          // Xn-1
        iSTLM20DD9F_EMA_K                               // Exponential smoothing factor
};
#elif   iSTLM20DD9F_ma == iWMA                          // Weighted moving average filter
static float32 STLM20DD9F_wma_buf[iSTLM20DD9F_WMA_num];
static const wma_f32_t STLM20DD9F_Phy_WMA =
{
        iInitial ,                                      // Initial state
        iSTLM20DD9F_WMA_num ,                          // Weighted moving average number & buf size
        0U ,                                            // buffer poition
        iSTLM20DD9F_WMA_num * (iSTLM20DD9F_WMA_num + 1)/2 ,       // kn sum
        &STLM20DD9F_wma_buf[0]                          // Xn buffer
};
#else                                                   // Non-moving average filter
#endif

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

```
const tbl_adc_t tbl_STLM20DD9F =
{
        iSTLM20DD9F             ,
        iSTLM20DD9F_pin         ,
        iSTLM20DD9F_xoff        ,
        iSTLM20DD9F_yoff        ,
        iSTLM20DD9F_gain        ,
        iSTLM20DD9F_max         ,
        iSTLM20DD9F_min         ,
        iSTLM20DD9F_ma          ,

#if     iSTLM20DD9F_ma == iSMA                  // Simple moving average filter
        &STLM20DD9F_Phy_SMA     ,
        (ema_f32_t*)iDummy_adr ,
        (wma_f32_t*)iDummy_adr
#elif   iSTLM20DD9F_ma == iEMA                  // Exponential moving average filter
        (sma_f32_t*)iDummy_adr ,
        &STLM20DD9F_Phy_EMA     ,
        (wma_f32_t*)iDummy_adr
#elif   iSTLM20DD9F_ma == iWMA                  // Weighted moving average filter
        (sma_f32_t*)iDummy_adr ,
        (ema_f32_t*)iDummy_adr ,
        &STLM20DD9F_Phy_WMA
#else                                           // Non-moving average filter
        (sma_f32_t*)iDummy_adr ,
        (ema_f32_t*)iDummy_adr ,
        (wma_f32_t*)iDummy_adr
#endif

};
```