

Component manufacturer

Specification document of STLM20DD9F

Model number	STLM20DD9F	
Datasheets	Illtra-low current 2.4 V precision analog temperature ser	200

STMicroelectronics

Datasheets Ultra-low current 2.4 V precision analog temperature sensor

(st.com)

01.00.00Oct 20,2022 Specification Ver New release Rui Long Lab Inc. https://rui-long-lab.com/ Documentation provided

1.	Component datasheet	2
	Component Software IF specification	
۷.	Component Software in Specification	
3	File Structure and Definitions	5

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://ossec.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

Temperature accuracy ± 1.5 ° C maximum temperature accuracy

at 25 $^{\circ}$ C ($\pm 0.5 ^{\circ}$ C typical)

Temperature range $-40 \text{ to } +85^{\circ} \text{ C}$

Range of power supply voltage (Vdd) 2.4 to 5.5[V]

Output voltage (Vout) Linear $-11.67 \text{ [mV/}^{\circ} \text{ C]} (-40 \text{ to } +85^{\circ} \text{ C})$ Calculation $\text{Vout} = 1.8583\text{V} + (-0.01167 \text{ V/}^{\circ} \text{ C} \times \text{Ta})$

 $Ta = (Vout - 1.8583V) / (-0.01167 V/^{\circ} C)$

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

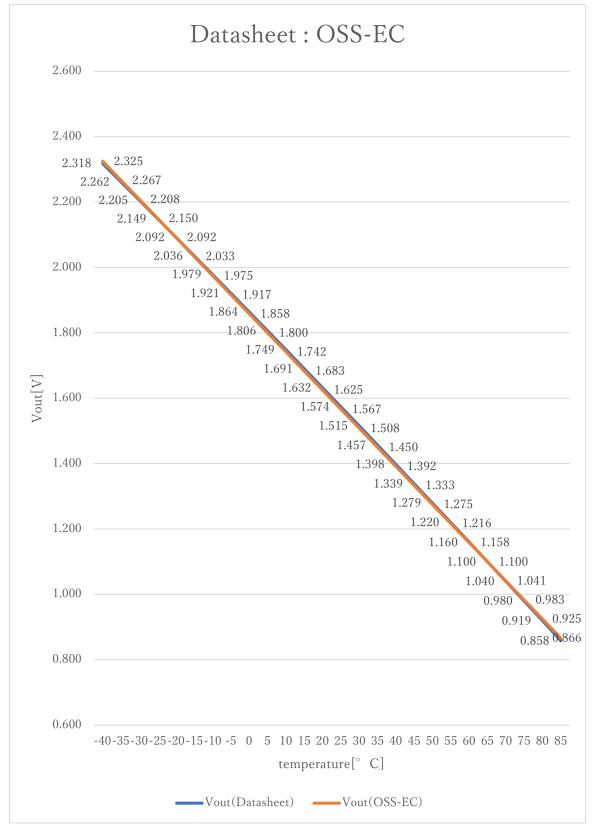
$$vi = (ai \times iADC_vdd) / 2^{iADC_bit}$$
 [V]

Voltage value to physical value conversion formula

```
y = (vi - iSTLM20DD9F\_xoff) / iSTLM20DD9F\_gain + iSTLM20DD9F\_yoff [°C] iSTLM20DD9F\_min <math>\leq y \leq iSTLM20DD9F\_max
```

```
A/D conversion value
ai
٧i
                 Sensor output voltage value [V]
i ADC vdd
                 Sensor supply voltage value [V]
iADC_bit
                 A/D conversion bit length
                 Temperature value [°C]
#define iSTLM20DD9F_xoff
                                                     // X offset [V]
                                   1.8583F
#define iSTLM20DD9F_yoff
                                   <u>0. 0F</u>
                                                     // Y offset [°C]
#define iSTLM20DD9F_gain
                                                     // Gain [V/°C]
                                   -0. 01167F
#define iSTLM20DD9F_max
                                                     // Temperature Max [°C]
                                   85. OF
#define iSTLM20DD9F_min
                                   -40. OF
                                                     // Temperature Min [°C]
```





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



3. File Structure and Definitions

STLM20DD9F.h

```
#include "user_define.h"
// Components number
#define iSTLM20DD9F
                                                          // STMicroelectronics STLM20DD9F
                             123U
// STLM20DD9F System Parts definitions
#define iSTLM20DD9F_xoff
                             1.8583F
                                                          // X offset [V]
#define iSTLM20DD9F_yoff
                             <u>0. 0F</u>
                                                          // Y offset [°C]
#define iSTLM20DD9F_gain
                             <u>-0. 01167F</u>
                                                          // Gain [V/°C]
#define iSTLM20DD9F_max
                             85. OF
                                                          // Temperature Max [°C]
#define iSTLM20DD9F_min
                             <u>-40. 0F</u>
                                                          // Temperature Min [°C]
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
        OU ,
                                                        // 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
        OU ,
                                                        // 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
        iSTLM2ODD9F_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
};
```