Specification document of MCP9701, MCP9701A

Component manufacturer	Microchip Technology		
Model number	MCP9701, MCP9701A		
Datasheets	Low-Power Linear Active Thermistor ICs (microchip.com)		
Specification Ver	01.00.00	Oct 13,2022	New release
	01.00.01	Oct 18,2022	Corrected license content
Documentation provided	Rui Long Lab Inc. https://rui-long-lab.com/		

1. Component datasheet

Temperature accuracy	±4.0° C (Max, 0 to +70° C) MCP9701
	±2.0° C (Max, 0 to +70° C) MCP9701A
Temperature range	-10 to +125° C
Range of power supply voltage (Vdd)	3.1 to 5.5[V]
Output voltage (Vout)	Linear 0.0195 [mV/° C] Typ.
	0 [° C] 0.4 [V] Typ.
Calculation	Vout = 0.4V + (0.0195 V/° C × Ta)
	Ta = (Vout – 0.4V) / (0.0195 V/° C)

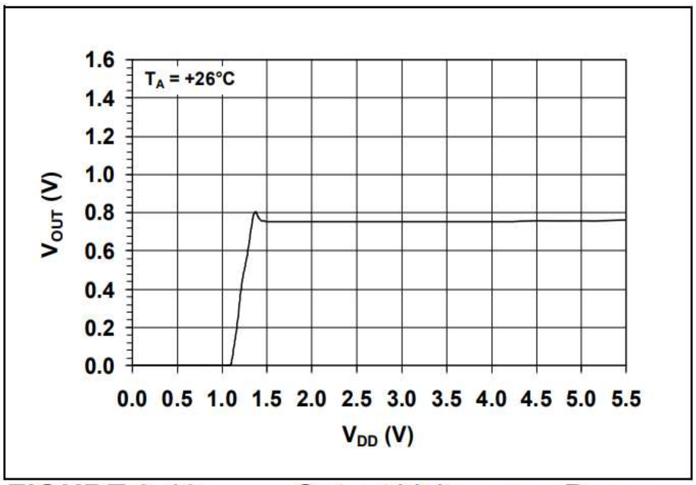


FIGURE 2-13: *Output Voltage vs. Power Supply.*

Applications	IoT etc

- Hard Disk Drives and Other PC Peripherals
- Entertainment Systems
- Home Appliance
- Office Equipment
- Battery Packs and Portable Equipment
- General Purpose Temperature Monitoring

2. Component Software IF specification

The software interface specifications based on the MCP9701, MCP9701A 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

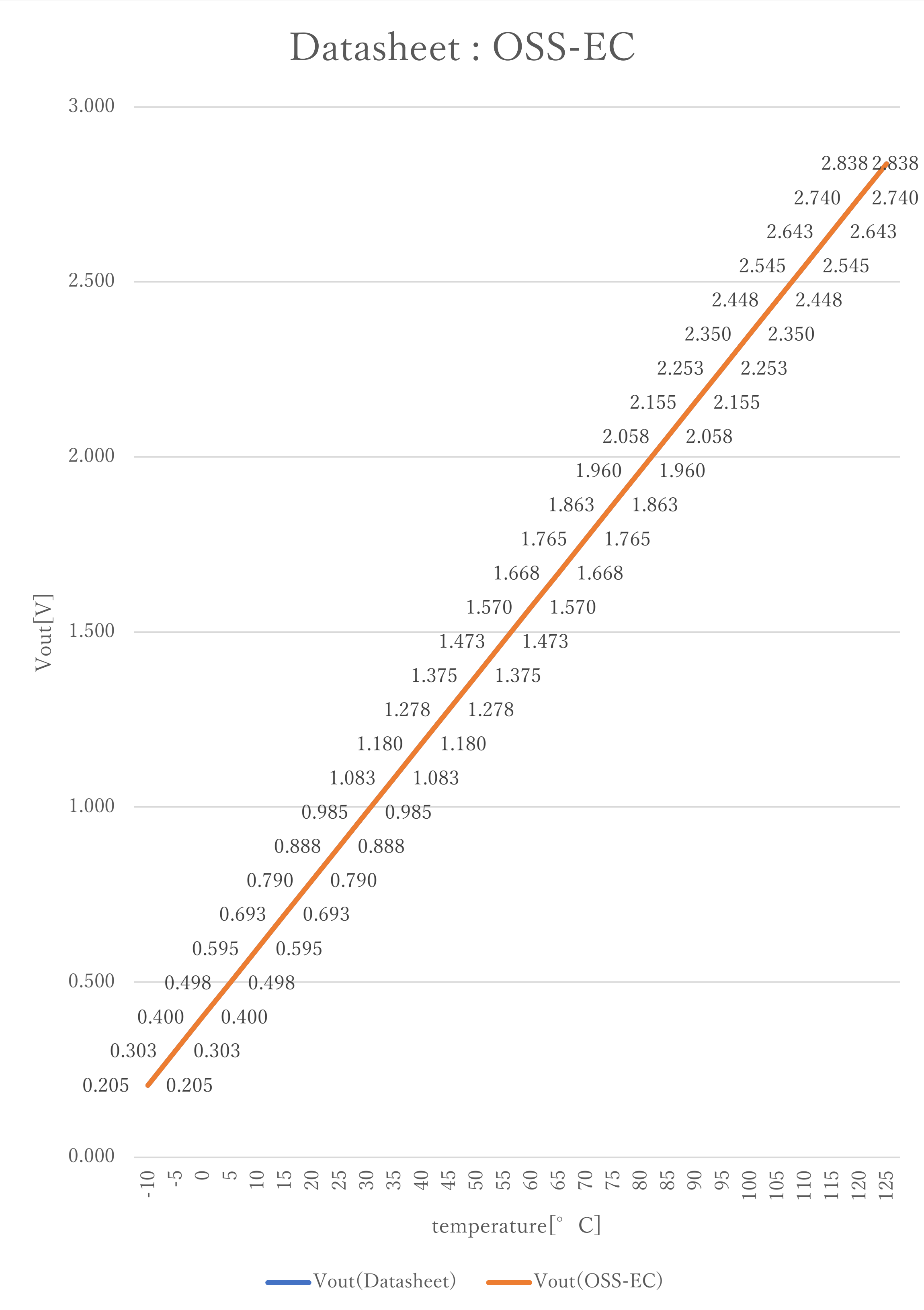
$$vi = (ai \times iADC\_vdd) / 2^{iADC\_bit} \quad [V]$$

Voltage value to physical value conversion formula

$$y = (vi - iMCP9701\_xoff) / iMCP9701\_gain + iMCP9701\_yoff \quad [℃]$$

$$iMCP9701\_min \leqq y \leqq iMCP9701\_max$$

```
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 iMCP9701_xoff       0.4F        // X offset [V]
#define iMCP9701_yoff       0.0F        // Y offset [℃]
#define iMCP9701_gain       0.0195F     // Gain [V/℃]
#define iMCP9701_max        125.0F      // Temperature Max [℃]
#define iMCP9701_min        -10.0F      // Temperature Min [℃]
```


Datasheet : OSS-EC
3.000
2.500
2.000
1.500
1.000
0.500
0.000
Vout[V]
0.205 0.205
0.303 0.303
0.400 0.400
0.498 0.498
0.595 0.595
0.693 0.693
0.790 0.790
0.888 0.888
0.985 0.985
1.083 1.083
1.180 1.180
1.278 1.278
1.375 1.375
1.473 1.473
1.570 1.570
1.668 1.668
1.765 1.765
1.863 1.863
1.960 1.960
2.058 2.058
2.155 2.155
2.253 2.253
2.350 2.350
2.448 2.448
2.545 2.545
2.643 2.643
2.740 2.740
2.838 2.838
-10 -5 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125
temperature[° C]
Vout(Datasheet)
Vout(OSS-EC)

3. File Structure and Definitions

MCP9701.h

```
#include "user_define.h"

// Components number
#define iMCP9701          115U                    // Microchip Technology MCP9701, MCP9701A

// MCP9701 System Parts definitions
#define iMCP9701_xoff     0.4F                    // X offset [V]
#define iMCP9701_yoff     0.0F                    // Y offset [℃]
#define iMCP9701_gain     0.0195F                 // Gain [V/℃]
#define iMCP9701_max      125.0F                  // Temperature Max [℃]
#define iMCP9701_min      -10.0F                  // Temperature Min [℃]

extern const tbl_adc_t tbl_MCP9701;
```

MCP9701.cpp

```
#include        "MCP9701.h"
#if     iMCP9701_ma == iSMA                         // Simple moving average filter
static float32 MCP9701_sma_buf[iMCP9701_SMA_num];
static const sma_f32_t MCP9701_Phy_SMA =
{
        iInitial ,                                  // Initial state
        iMCP9701_SMA_num ,                      // Simple moving average number & buf size
        0U ,                                        // buffer position
        0.0F ,                                      // sum
        &MCP9701_sma_buf[0]                         // buffer
};
#elif   iMCP9701_ma == iEMA                         // Exponential moving average filter
static const ema_f32_t MCP9701_Phy_EMA =
{
        iInitial ,                                  // Initial state
        0.0F ,                                      // Xn-1
        iMCP9701_EMA_K                              // Exponential smoothing factor
};
#elif   iMCP9701_ma == iWMA                         // Weighted moving average filter
static float32 MCP9701_wma_buf[iMCP9701_WMA_num];
static const wma_f32_t MCP9701_Phy_WMA =
{
        iInitial ,                                  // Initial state
        iMCP9701_WMA_num ,                    // Weighted moving average number & buf size
        0U ,                                        // buffer poition
        iMCP9701_WMA_num * (iMCP9701_WMA_num + 1)/2 ,   // kn sum
        &MCP9701_wma_buf[0]                         // Xn buffer
};
#else                                               // Non-moving average filter
#endif

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

```
const tbl_adc_t tbl_MCP9701 =
{
        iMCP9701                ,
        iMCP9701_pin            ,
        iMCP9701_xoff           ,
        iMCP9701_yoff           ,
        iMCP9701_gain           ,
        iMCP9701_max            ,
        iMCP9701_min            ,
        iMCP9701_ma             ,

#if     iMCP9701_ma == iSMA                      // Simple moving average filter
        &MCP9701_Phy_SMA        ,
        (ema_f32_t*)iDummy_adr  ,
        (wma_f32_t*)iDummy_adr
#elif   iMCP9701_ma == iEMA                      // Exponential moving average filter
        (sma_f32_t*)iDummy_adr  ,
        &MCP9701_Phy_EMA        ,
        (wma_f32_t*)iDummy_adr
#elif   iMCP9701_ma == iWMA                      // Weighted moving average filter
        (sma_f32_t*)iDummy_adr  ,
        (ema_f32_t*)iDummy_adr  ,
        &MCP9701_Phy_WMA
#else                                            // Non-moving average filter
        (sma_f32_t*)iDummy_adr  ,
        (ema_f32_t*)iDummy_adr  ,
        (wma_f32_t*)iDummy_adr
#endif

};
```