# Specification document of MPX5999D

| | |
|---|---|
| Component manufacturer | NXP Semiconductors |
| Model number | MPX5999D |
| Datasheets | MPX5999D Integrated Silicon Pressure Sensor On-Chip Signal Conditioned, Temperature Compensated and Calibrated - Data sheet (nxp.com) |
| Specification Ver | 01.00.00 Oct 18,2022 New release |
| Documentation provided | Rui Long Lab Inc. https://rui-long-lab.com/ |

## 1. Component datasheet

| | |
|---|---|
| Pressure range | 0 to 1000[kPa] |
| Range of power supply voltage( Vdd ) | 4.75 to 5.25[V]　5.0[V]Typ. |
| Output voltage ( Vout ) | Vout = Vdd × ( P × 0.000901 +0.04 ) ± Error<br>Vdd =5.0[V]<br>Temperature 0 to 85° C<br>P = (( Vout / Vdd ) – 0.04 ) / 0.000901 |
| Vdd vs Vout | link |

| | |
|---|---|
| Applications | IoT etc<br>・ Ideally suited for microprocessor or microcontroller-based systems |

## 2. Component Software IF specification

The software interface specifications based on the MPX5999D 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

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

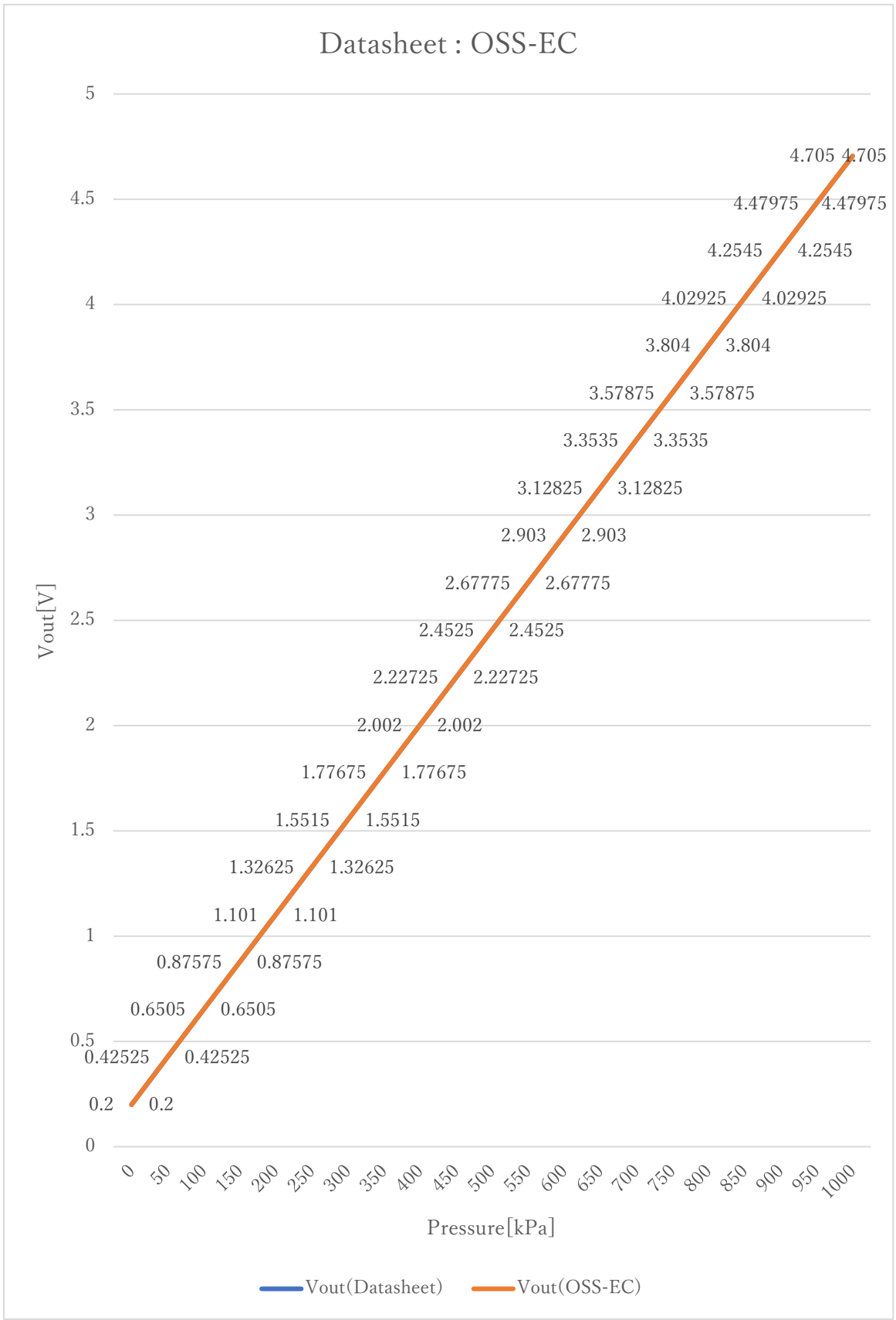
Voltage value to physical value conversion formula

$$y = ( vi - iMPX5999D\_xoff ) / iMPX5999D\_gain + iMPX5999D\_yoff \quad [kPa]$$

$$iMPX5999D\_min \leqq y \leqq iMPX5999D\_max$$

| | |
|---|---|
| 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 | Pressure value [kPa] |

```
#define iMPX5999D_xoff          ( 0.04F*iADC_vdd )      // X offset [V]
#define iMPX5999D_yoff          0.0F                    // Y offset [kPa]
#define iMPX5999D_gain          ( 0.000901F*iADC_vdd )  // Gain [V/kPa]
#define iMPX5999D_max           1000.0F                 // Pressure Max [kPa]
#define iMPX5999D_min           0.0F                    // Pressure Min [kPa]
```


Datasheet : OSS-EC
Vout[V]
Pressure[kPa]
Vout(Datasheet)
Vout(OSS-EC)
0
0.2
0.42525
0.6505
0.87575
1.101
1.32625
1.5515
1.77675
2.002
2.22725
2.4525
2.67775
2.903
3.12825
3.3535
3.57875
3.804
4.02925
4.2545
4.47975
4.705

## 3. File Structure and Definitions

MPX5999D.h

```
#include "user_define.h"

// Components number
#define iMPX5999D               118U                    // NXP MPX5999D

// MPX5999D System Parts definitions
#define iMPX5999D_xoff          ( 0.04F*iADC_vdd )      // X offset [V]
#define iMPX5999D_yoff          0.0F                    // Y offset [kPa]
#define iMPX5999D_gain          ( 0.000901F*iADC_vdd )  // Gain [V/kPa]
#define iMPX5999D_max           1000.0F                 // Pressure Max [kPa]
#define iMPX5999D_min           0.0F                    // Pressure Min [kPa]

extern const tbl_adc_t tbl_MPX5999D;
```

MPX5999D.cpp

```
#include        "MPX5999D.h"
#if     iMPX5999D_ma == iSMA                            // Simple moving average filter
static float32 MPX5999D_sma_buf[iMPX5999D_SMA_num];
static const sma_f32_t MPX5999D_Phy_SMA =
{
        iInitial ,                                      // Initial state
        iMPX5999D_SMA_num ,                            // Simple moving average number & buf size
        0U ,                                            // buffer position
        0.0F ,                                          // sum
        &MPX5999D_sma_buf[0]                            // buffer
};
#elif   iMPX5999D_ma == iEMA                            // Exponential moving average filter
static const ema_f32_t MPX5999D_Phy_EMA =
{
        iInitial ,                                      // Initial state
        0.0F ,                                          // Xn-1
        iMPX5999D_EMA_K                                 // Exponential smoothing factor
};
#elif   iMPX5999D_ma == iWMA                            // Weighted moving average filter
static float32 MPX5999D_wma_buf[iMPX5999D_WMA_num];
static const wma_f32_t MPX5999D_Phy_WMA =
{
        iInitial ,                                      // Initial state
        iMPX5999D_WMA_num ,                            // Weighted moving average number & buf size
        0U ,                                            // buffer poition
        iMPX5999D_WMA_num * (iMPX5999D_WMA_num + 1)/2 , // kn sum
        &MPX5999D_wma_buf[0]                            // Xn buffer
};
#else                                                   // Non-moving average filter
#endif

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

```
const tbl_adc_t tbl_MPX5999D =
{
        iMPX5999D               ,
        iMPX5999D_pin           ,
        iMPX5999D_xoff          ,
        iMPX5999D_yoff          ,
        iMPX5999D_gain          ,
        iMPX5999D_max           ,
        iMPX5999D_min           ,
        iMPX5999D_ma            ,

#if     iMPX5999D_ma == iSMA                    // Simple moving average filter
        &MPX5999D_Phy_SMA       ,
        (ema_f32_t*)iDummy_adr  ,
        (wma_f32_t*)iDummy_adr
#elif   iMPX5999D_ma == iEMA                    // Exponential moving average filter
        (sma_f32_t*)iDummy_adr  ,
        &MPX5999D_Phy_EMA       ,
        (wma_f32_t*)iDummy_adr
#elif   iMPX5999D_ma == iWMA                    // Weighted moving average filter
        (sma_f32_t*)iDummy_adr  ,
        (ema_f32_t*)iDummy_adr  ,
        &MPX5999D_Phy_WMA
#else                                            // Non-moving average filter
        (sma_f32_t*)iDummy_adr  ,
        (ema_f32_t*)iDummy_adr  ,
        (wma_f32_t*)iDummy_adr
#endif

};
```