Specification document of MPXHZ6250A

Component manufacturer	NXP Semiconductors
Model number	MPXHZ6250A
Datasheets	MPXHZ6250A, Media Resistant and High Temperature Accuracy Integrated Silicon Pressure Sensor for Measuring Absolute Pressure, On-Chip Signal Conditioned, Temperature Compensated and Calibrated (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	20 to 250[kPa] 1.5% maximum error 0 to 85° C
Range of power supply voltage(Vdd)	4.75 to 5.25[V] 5.0[V]Typ.

Output voltage (Vout)	Vout = Vdd × (P × 0.0040 - 0.040) ± Error
	Vdd =5.0[V]
	Temperature 0 to 85° C
	P = ((Vout / Vdd) + 0.04) / 0.004

Vdd vs Vout	link

Applications

IoT etc

- Industrial controls

Automotive

- Engine Control/Liquified Petroleum Gas (LPG)

2. Component Software IF specification

The software interface specifications based on the MPXHZ6250A 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

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

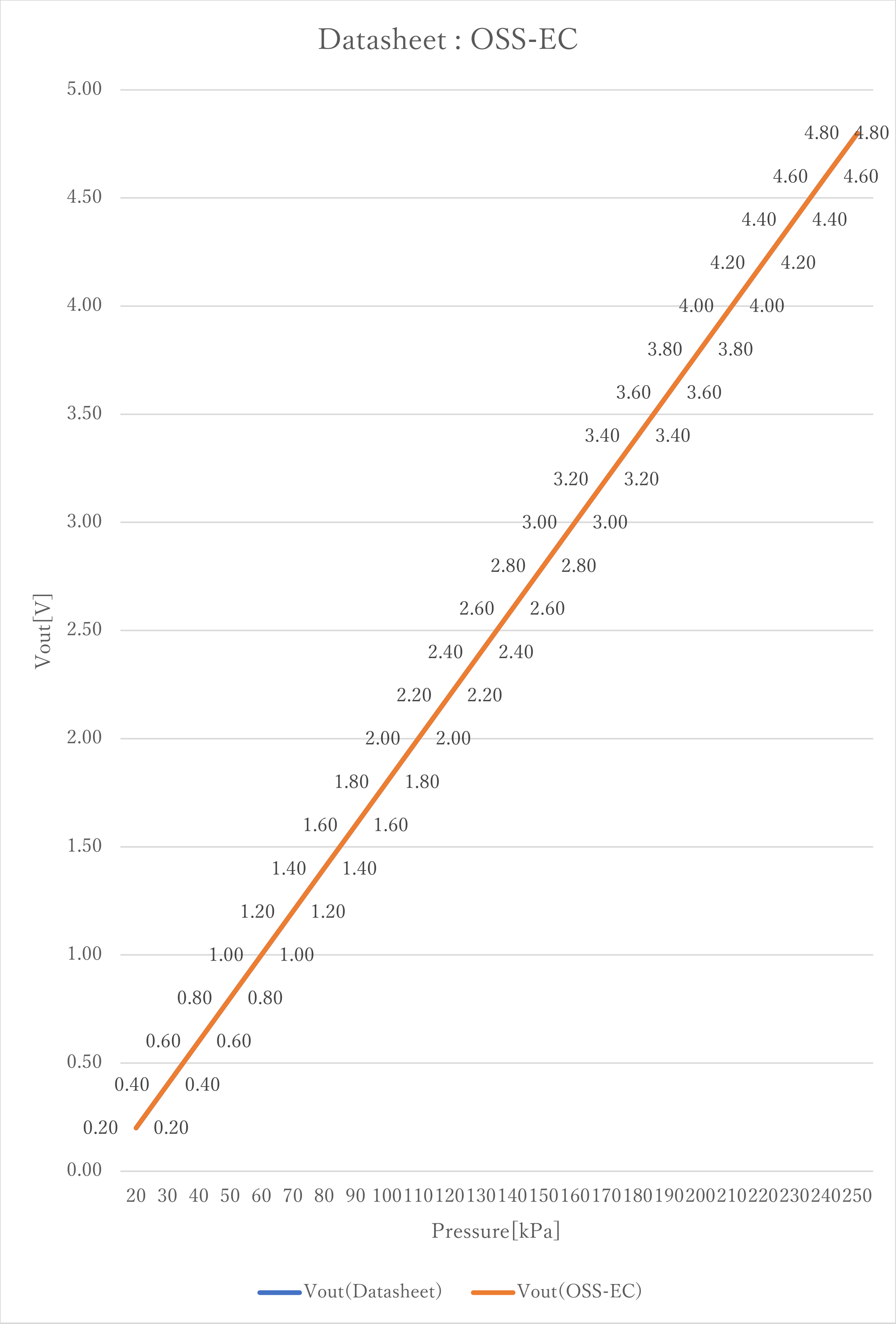
Voltage value to physical value conversion formula

$$y = (vi - iMPXHZ6250A\_xoff) / iMPXHZ6250A\_gain + iMPXHZ6250A\_yoff \quad [kPa]$$

$$iMPXHZ6250A\_min \leqq y \leqq iMPXHZ6250A\_max$$

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 iMPXHZ6250A_xoff        ( -0.040F*iADC_vdd )    // X offset [V]
#define iMPXHZ6250A_yoff        0.0F                    // Y offset [kPa]
#define iMPXHZ6250A_gain        ( 0.0040F*iADC_vdd )    // Gain [V/kPa]
#define iMPXHZ6250A_max         250.0F                  // Pressure Max [kPa]
#define iMPXHZ6250A_min         20.0F                   // Pressure Min [kPa]
```


Datasheet : OSS-EC
5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
Vout[V]
4.80 4.80
4.60 4.60
4.40 4.40
4.20 4.20
4.00 4.00
3.80 3.80
3.60 3.60
3.40 3.40
3.20 3.20
3.00 3.00
2.80 2.80
2.60 2.60
2.40 2.40
2.20 2.20
2.00 2.00
1.80 1.80
1.60 1.60
1.40 1.40
1.20 1.20
1.00 1.00
0.80 0.80
0.60 0.60
0.40 0.40
0.20 0.20
20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250
Pressure[kPa]
Vout(Datasheet)
Vout(OSS-EC)

3. File Structure and Definitions

MPXHZ6250A.h

```
#include "user_define.h"

// Components number
#define iMPXHZ6250A         120U                        // NXP MPXHZ6250A

// MPXHZ6250A System Parts definitions
#define iMPXHZ6250A_xoff    ( -0.040F*iADC_vdd )        // X offset [V]
#define iMPXHZ6250A_yoff    0.0F                        // Y offset [kPa]
#define iMPXHZ6250A_gain    ( 0.0040F*iADC_vdd )        // Gain [V/kPa]
#define iMPXHZ6250A_max     250.0F                      // Pressure Max [kPa]
#define iMPXHZ6250A_min     20.0F                       // Pressure Min [kPa]

extern const tbl_adc_t tbl_MPXHZ6250A;
```

MPXHZ6250A.cpp

```
#include        ”MPXHZ6250A.h”
#if     iMPXHZ6250A_ma == iSMA                         // Simple moving average filter
static float32 MPXHZ6250A_sma_buf[iMPXHZ6250A_SMA_num];
static const sma_f32_t MPXHZ6250A_Phy_SMA =
{
        iInitial ,                                     // Initial state
        iMPXHZ6250A_SMA_num ,                         // Simple moving average number & buf size
        0U ,                                           // buffer position
        0.0F ,                                         // sum
        &MPXHZ6250A_sma_buf[0]                         // buffer
};
#elif   iMPXHZ6250A_ma == iEMA                         // Exponential moving average filter
static const ema_f32_t MPXHZ6250A_Phy_EMA =
{
        iInitial ,                                     // Initial state
        0.0F ,                                         // Xn-1
        iMPXHZ6250A_EMA_K                              // Exponential smoothing factor
};
#elif   iMPXHZ6250A_ma == iWMA                         // Weighted moving average filter
static float32 MPXH6115A_wma_buf[iMPXHZ6250A_WMA_num];
static const wma_f32_t MPXHZ6250A_Phy_WMA =
{
        iInitial ,                                     // Initial state
        iMPXHZ6250A_WMA_num ,                        // Weighted moving average number & buf size
        0U ,                                           // buffer poition
        iMPXHZ6250A_WMA_num * (iMPXHZ6250A_WMA_num + 1)/2 , // kn sum
        &MPXHZ6250A_wma_buf[0]                         // Xn buffer
};
#else                                                  // Non-moving average filter
#endif

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

```
const tbl_adc_t tbl_MPXHZ6250A =
{
        iMPXHZ6250A             ,
        iMPXHZ6250A_pin         ,
        iMPXHZ6250A_xoff        ,
        iMPXHZ6250A_yoff        ,
        iMPXHZ6250A_gain        ,
        iMPXHZ6250A_max         ,
        iMPXHZ6250A_min         ,
        iMPXHZ6250A_ma          ,

#if     iMPXHZ6250A_ma == iSMA                          // Simple moving average filter
        &MPXHZ6250A_Phy_SMA     ,
        (ema_f32_t*)iDummy_adr  ,
        (wma_f32_t*)iDummy_adr
#elif   iMPXHZ6250A_ma == iEMA                          // Exponential moving average filter
        (sma_f32_t*)iDummy_adr  ,
        &MPXHZ6250A_Phy_EMA     ,
        (wma_f32_t*)iDummy_adr
#elif   iMPXHZ6250A_ma == iWMA                          // Weighted moving average filter
        (sma_f32_t*)iDummy_adr  ,
        (ema_f32_t*)iDummy_adr  ,
        &MPXHZ6250A_Phy_WMA
#else                                                   // Non-moving average filter
        (sma_f32_t*)iDummy_adr  ,
        (ema_f32_t*)iDummy_adr  ,
        (wma_f32_t*)iDummy_adr
#endif

};
```