



## Specification document of MPXA4250A

Component manufacturer	NXP Semiconductors		
Model number	MPXA4250A		
Datasheets	<a href="https://www.nxp.com/docs/en/data-sheet/MPX4250A.pdf">https://www.nxp.com/docs/en/data-sheet/MPX4250A.pdf</a>		
Specification Ver	01.00.00	Aug 31,2022	New release
	01.00.01	Sep 10,2022	Corrected license URL
Documentation provided	Rui Long Lab Inc. <a href="https://rui-long-lab.com/">https://rui-long-lab.com/</a>		

### 1. Component Software IF specification ..... 2

#### 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://oss-ec.com/license\\_agreement/](https://oss-ec.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. We reserve the right to change these Terms of Use at any time and for any reason. We strongly recommend that you review the Terms of Use periodically.

## 1. Component Software IF specification

The software interface specifications based on the MPXA4250A 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

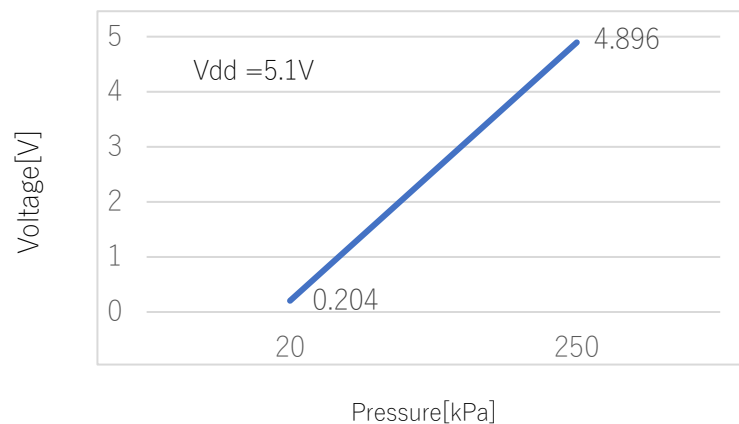
$$v_i = ( a_i \times i_{ADC\_vdd} ) / 2^{i_{ADC\_bit}} \quad [V]$$

Voltage value to physical value conversion formula

$$y = ( v_i - i_{MPXA4250A\_xoff} ) / i_{MPXA4250A\_gain} + i_{MPXA4250A\_yoff} \quad [kPa]$$

$$i_{MPXA4250A\_min} \leq y \leq i_{MPXA4250A\_max}$$

$a_i$	A/D conversion value	
$v_i$	Sensor output voltage value [V]	
$i_{ADC\_vdd}$	Sensor supply voltage value [V]	
$i_{ADC\_bit}$	A/D conversion bit length	
$y$	Pressure value [kPa]	
#define $i_{MPXA4250A\_xoff}$	$( -0.04F \times i_{ADC\_vdd} )$	// X offset [V]
#define $i_{MPXA4250A\_yoff}$	$0.0F$	// Y offset [kPa]
#define $i_{MPXA4250A\_gain}$	$( 0.004F \times i_{ADC\_vdd} )$	// Gain [V/kPa]
#define $i_{MPXA4250A\_max}$	$250.0F$	// Pressure Max [kPa]
#define $i_{MPXA4250A\_min}$	$20.0F$	// Pressure Min [kPa]



## 2. File Structure and Definitions

### MPXA4250A.h

```
#include "user_define.h"

// Components number
#define IMPXA4250A          100U          // NXP MPXA4250A

// MPXA4250A System Parts definitions
#define IMPXA4250A_xoff      ( -0.04F*iADC_vdd )    // X offset [V]
#define IMPXA4250A_yoff      0.0F                  // Y offset [kPa]
#define IMPXA4250A_gain      ( 0.004F*iADC_vdd )    // Gain [V/kPa]
#define IMPXA4250A_max        250.0F                // Pressure Max [kPa]
#define IMPXA4250A_min        20.0F                 // Pressure Min [kPa]

extern const tbl_adc_t tbl_MPX4250A;
```

## MPXA4250A.cpp

```
#include      "MPXA4250A.h"

#if      iMPXA4250A_ma == iSMA                                // Simple moving average filter
static float32 MPXA4250A_sma_buf[iMPXA4250A_SMA_num];
static const sma_f32_t MPXA4250A_Phy_SMA =
{
    iInitial ,                                                // Initial state
    iMPXA4250A_SMA_num ,                                       // Simple moving average number & buf
size
    0U ,                                                        // buffer position
    0.0F ,                                                      // sum
    &MPXA4250A_sma_buf[0]                                       // buffer
};

#elif      iMPXA4250A_ma == iEMA                                // Exponential moving average filter
static const ema_f32_t MPXA4250A_Phy_EMA =
{
    iInitial ,                                                // Initial state
    0.0F ,                                                      // Xn-1
    iMPXA4250A_EMA_K                                           // Exponential smoothing factor
};

#elif      iMPXA4250A_ma == iWMA                                // Weighted moving average filter
static float32 MPXA4250A_wma_buf[iMPXA4250A_WMA_num];
static const wma_f32_t MPXA4250A_Phy_WMA =
{
    iInitial ,                                                // Initial state
    iMPXA4250A_WMA_num ,                                       // Weighted moving average number & buf
size
    0U ,                                                        // buffer position
    iMPXA4250A_WMA_num * (iMPXA4250A_WMA_num + 1)/2 , // kn sum
    &MPXA4250A_wma_buf[0]                                       // Xn buffer
};

#else                                                        // Non-moving average filter
#endif

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

```
const tbl_adc_t tbl_MPXA4250A =
{
    iMPXA4250A          ,
    iMPXA4250A_pin      ,
    iMPXA4250A_xoff     ,
    iMPXA4250A_yoff     ,
    iMPXA4250A_gain     ,
    iMPXA4250A_max      ,
    iMPXA4250A_min      ,
    iMPXA4250A_ma       ,

    #if iMPXA4250A_ma == iSMA // Simple moving average filter
        &MPXA4250A_Phy_SMA   ,
        iDummy_adr          ,
        iDummy_adr
    #elif iMPXA4250A_ma == iEMA // Exponential moving average filter
        iDummy_adr          ,
        &MPXA4250A_Phy_EMA   ,
        iDummy_adr
    #elif iMPXA4250A_ma == iWMA // Weighted moving average filter
        iDummy_adr          ,
        iDummy_adr          ,
        &MPXA4250A_Phy_WMA
    #else // Non-moving average filter
        iDummy_adr          ,
        iDummy_adr          ,
        iDummy_adr
    #endif

};
```