Specification document of CHS-UPS, CHS-UPR, CHS-UGS, CHS-UGR

Component manufacturer	TDK		
Model number	CHS-UPS, CHS-UPR, CHS-UGS, CHS-UGR		
Datasheets	sensor_humidity_chs_en.pdf (tdk.com)		
Specification Ver	01.00.00	Oct 28,2022	New release
Documentation provided	Rui Long Lab Inc. https://rui-long-lab.com/		

1. Component datasheet

Humidity accuracy	CHS-UPS,-UPR ±3%RH Edc=5V, 25° C, 5 to 95%RH
	CHS-UGS,-UGR ±5%RH Edc=5V, 25° C, 5 to 95%RH
Humidity range	5% to 95% RH
Range of power supply voltage (Vdd)	4.75 to 5.25[V]
	Standard : 5[V]
Output voltage (Vout)	Linear 10 [mV/%RH] Edc=5V, 25° C, 5 to 95%RH
Calculation	Vout = 0.0V + (0.01 V/%RH × H)
	H = (Vout – 0.0V) / (0.01 V/%RH)
Vdd vs Vout	Non-link

Applications IoT etc

- Refrigerators (condensation prevention)
- Air conditioners (indoor humidity control)
- PPCs, LBPs (image quality control)
- Industrial electronic humidity sensors, air conditioners for plant factories, etc.

2. Component Software IF specification

The software interface specifications based on the CHS-MSS 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

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

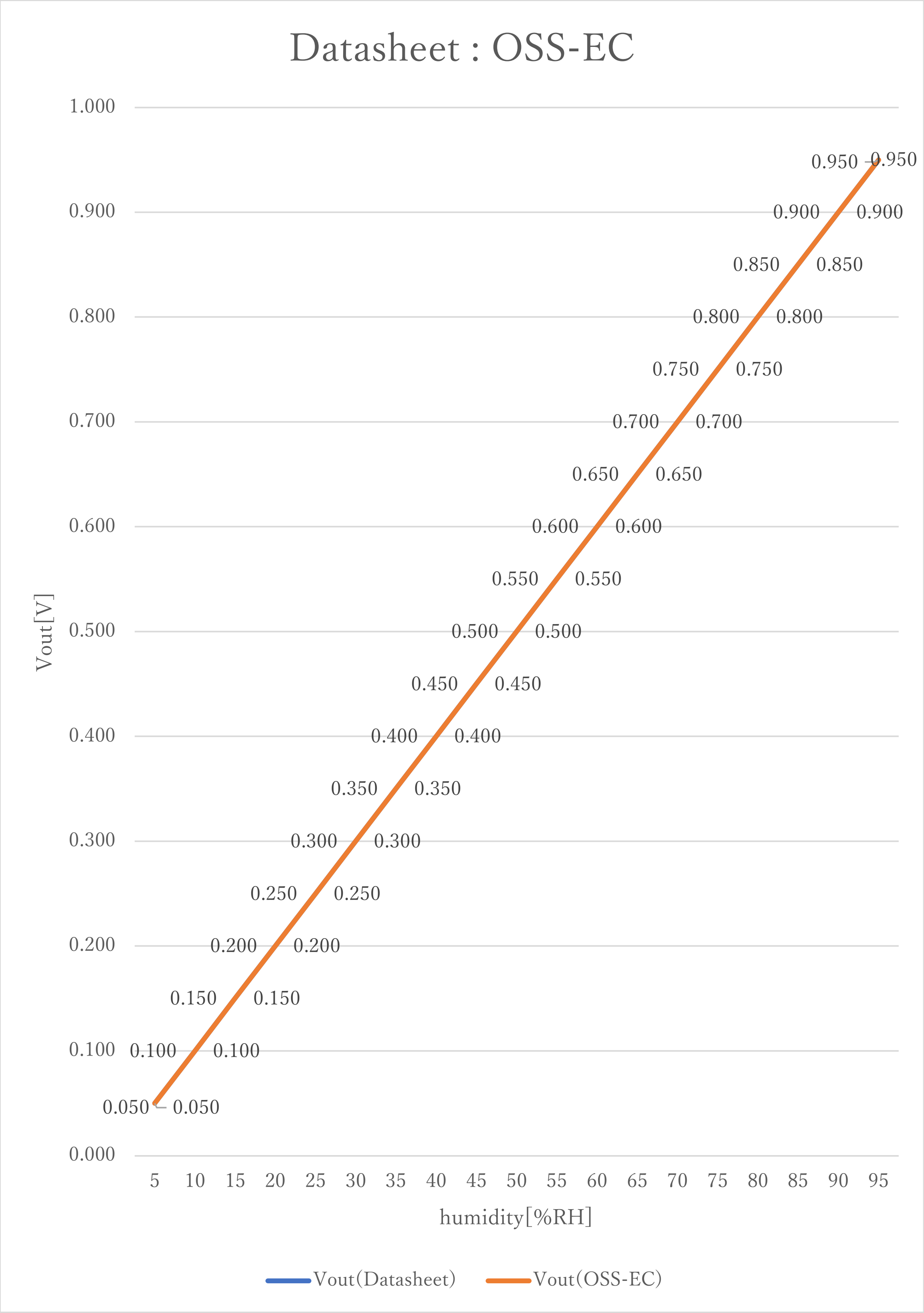
Voltage value to physical value conversion formula

$$y = (vi - iCHS\_UPS\_xoff) / iCHS\_UPS\_gain + iCHS\_UPS\_yoff \quad [\%RH]$$

$$iCHS\_UPS\_min \leqq y \leqq iCHS\_UPS\_max$$

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	Humidity value [%RH]

```
#define iCHS_UPS_xoff          0.0F          // X offset [V]
#define iCHS_UPS_yoff          0.0F          // Y offset [%RH]
#define iCHS_UPS_gain          0.01F         // Gain [V/%RH]
#define iCHS_UPS_max           95.0F         // Humidity Max [%RH]
#define iCHS_UPS_min           5.0F          // Humidity Min [%RH]
```


Datasheet : OSS-EC
Vout[V]
1.000
0.900
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
0.050 0.050
0.100 0.100
0.150 0.150
0.200 0.200
0.250 0.250
0.300 0.300
0.350 0.350
0.400 0.400
0.450 0.450
0.500 0.500
0.550 0.550
0.600 0.600
0.650 0.650
0.700 0.700
0.750 0.750
0.800 0.800
0.850 0.850
0.900 0.900
0.950 0.950
5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95
humidity[%RH]
Vout(Datasheet)
Vout(OSS-EC)

3. File Structure and Definitions

CHS_UPS.h

```
#include "user_define.h"

// Components number
#define iCHS_UPS            125U                        // TDK CHS-UPS, CHS-UPR, CHS-UGS, CHS-UGR

// CHS_MSS System Parts definitions
#define iCHS_UPS_xoff       0.0F                        // X offset [V]
#define iCHS_UPS_yoff       0.0F                        // Y offset [%RH]
#define iCHS_UPS_gain       0.01F                       // Gain [V/%RH]
#define iCHS_UPS_max        95.0F                       // Humidity Max [%RH]
#define iCHS_UPS_min        5.0F                        // Humidity Min [%RH]

extern const tbl_adc_t tbl_CHS_UPS;
```

CHS_UPS.cpp

```
#include        ″ CHS_UPS.h″
#if     iCHS_UPS_ma == iSMA                             // Simple moving average filter
static float32 CHS_UPS_sma_buf[iCHS_UPS_SMA_num];
static const sma_f32_t CHS_UPS_Phy_SMA =
{
        iInitial ,                                      // Initial state
        iCHS_UPS_SMA_num ,                         // Simple moving average number & buf size
        0U ,                                            // buffer position
        0.0F ,                                          // sum
        &CHS_UPS_sma_buf[0]                             // buffer
};
#elif   iCHS_UPS_ma == iEMA                             // Exponential moving average filter
static const ema_f32_t CHS_UPS_Phy_EMA =
{
        iInitial ,                                      // Initial state
        0.0F ,                                          // Xn-1
        iCHS_UPS_EMA_K                                  // Exponential smoothing factor
};
#elif   iCHS_UPS_ma == iWMA                             // Weighted moving average filter
static float32 CHS_UPS_wma_buf[iCHS_UPS_WMA_num];
static const wma_f32_t CHS_UPS_Phy_WMA =
{
        iInitial ,                                      // Initial state
        iCHS_UPS_WMA_num ,                       // Weighted moving average number & buf size
        0U ,                                            // buffer poition
        iCHS_UPS_WMA_num * (iCHS_UPS_WMA_num + 1)/2 ,   // kn sum
        &CHS_UPS_wma_buf[0]                             // Xn buffer
};
#else                                                   // Non-moving average filter
#endif

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

```
const tbl_adc_t tbl_CHS_UPS =
{
        iCHS_UPS                ,
        iCHS_UPS_pin            ,
        iCHS_UPS_xoff           ,
        iCHS_UPS_yoff           ,
        iCHS_UPS_gain           ,
        iCHS_UPS_max            ,
        iCHS_UPS_min            ,
        iCHS_UPS_ma             ,

#if     iCHS_UPS_ma == iSMA                        // Simple moving average filter
        &CHS_UPS_Phy_SMA        ,
        (ema_f32_t*)iDummy_adr ,
        (wma_f32_t*)iDummy_adr
#elif   iCHS_UPS_ma == iEMA                        // Exponential moving average filter
        (sma_f32_t*)iDummy_adr ,
        &CHS_UPS_Phy_EMA        ,
        (wma_f32_t*)iDummy_adr
#elif   iCHS_UPS_ma == iWMA                        // Weighted moving average filter
        (sma_f32_t*)iDummy_adr ,
        (ema_f32_t*)iDummy_adr ,
        &CHS_UPS_Phy_WMA
#else                                              // Non-moving average filter
        (sma_f32_t*)iDummy_adr ,
        (ema_f32_t*)iDummy_adr ,
        (wma_f32_t*)iDummy_adr
#endif

};
```