Specification document of BD1020HFV

Component manufacturer	ROHM Semiconductor
Model number	BD1020HFV
Datasheets	BD1020HFV : Sensors & MEMS (rohm.com)
Specification Ver	01.00.00 Oct 20,2022 New release
Documentation provided	Rui Long Lab Inc. https://rui-long-lab.com/

1. Component datasheet 2
2. Component Software IF specification 3
3. File Structure and Definitions 5

1. Component datasheet

Temperature accuracy	±1.5° C(Max) @Ta=30° C
	±2.5° C (Max) @Ta=-30° C, +100° C
Temperature range	-30 to +100° C
Range of power supply voltage (Vdd)	2.4 to 5.5[V]
Output voltage (Vout)	Linear -8.2 [mV/° C] Typ.
	30 [° C] 1.3 [V] Typ.
Calculation	Vout = 1.3V + (-0.0082 V/° C × (Ta - 30° C))
	Ta = (Vout – 1.3V) / (-0.0082 V/° C) + 30° C
Vdd vs Vout	Non-link

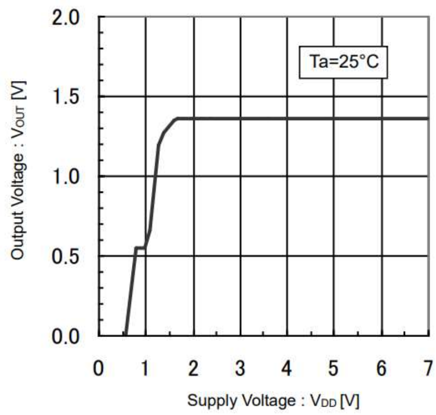


Figure 3. Output Voltage vs Supply Voltage

Applications IoT etc

- Cell Phone (RF Module, Battery Thermal Management)
- Audio Systems
- Digital Still Camera, LCD, PDP
- Optical pick up module for DVD and BlueRay

2. Component Software IF specification

The software interface specifications based on the BD1020HFV 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

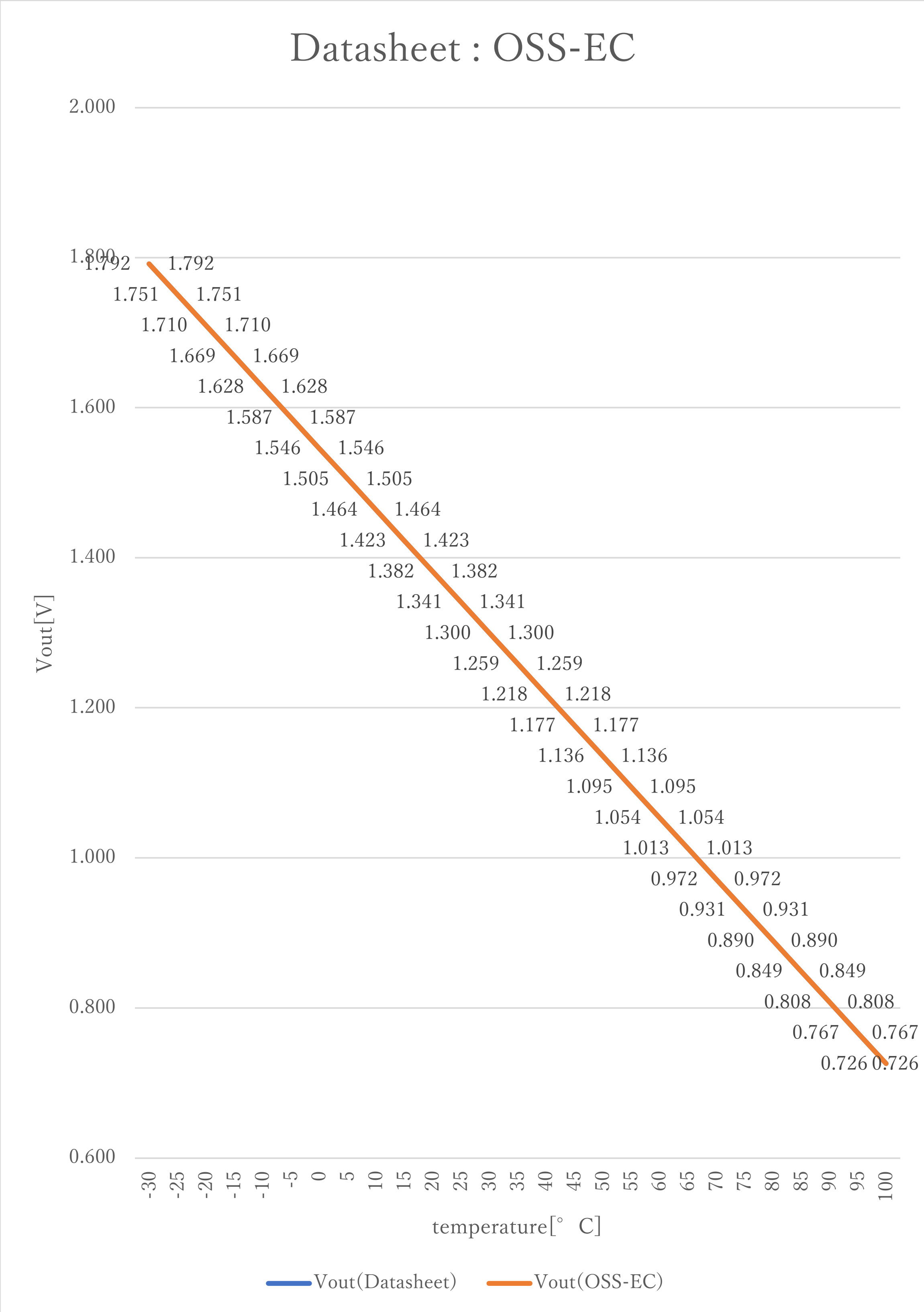
$$vi = (ai \times iADC\_vdd) / 2^{iADC\_bit} \quad [V]$$

Voltage value to physical value conversion formula

$$y = (vi - iBD1020HFV\_xoff) / iBD1020HFV\_gain + iBD1020HFV\_yoff \quad [℃]$$

$$iBD1020HFV\_min \leqq y \leqq iBD1020HFV\_max$$

```
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 iBD1020HFV_xoff         1.3F            // X offset [V]
#define iBD1020HFV_yoff         30.0F           // Y offset [℃]
#define iBD1020HFV_gain         -0.0082F        // Gain [V/℃]
#define iBD1020HFV_max          100.0F          // Temperature Max [℃]
#define iBD1020HFV_min          -30.0F          // Temperature Min [℃]
```


Datasheet : OSS-EC
2.000
1.800
1.600
1.400
1.200
1.000
0.800
0.600
Vout[V]
1.792 1.792
1.751 1.751
1.710 1.710
1.669 1.669
1.628 1.628
1.587 1.587
1.546 1.546
1.505 1.505
1.464 1.464
1.423 1.423
1.382 1.382
1.341 1.341
1.300 1.300
1.259 1.259
1.218 1.218
1.177 1.177
1.136 1.136
1.095 1.095
1.054 1.054
1.013 1.013
0.972 0.972
0.931 0.931
0.890 0.890
0.849 0.849
0.808 0.808
0.767 0.767
0.726 0.726
-30 -25 -20 -15 -10 -5 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
temperature[° C]
Vout(Datasheet)
Vout(OSS-EC)

3. File Structure and Definitions

BD1020HFV.h

```
#include "user_define.h"

// Components number
#define iBD1020HFV          122U                        // ROHM BD1020HFV

// BD1020HFV System Parts definitions
#define iBD1020HFV_xoff     1.3F                        // X offset [V]
#define iBD1020HFV_yoff     30.0F                       // Y offset [℃]
#define iBD1020HFV_gain     -0.0082F                    // Gain [V/℃]
#define iBD1020HFV_max      100.0F                      // Temperature Max [℃]
#define iBD1020HFV_min      -30.0F                      // Temperature Min [℃]

extern const tbl_adc_t tbl_BD1020HFV;
```

BD1020HFV.cpp

```
#include        "BD1020HFV.h"
#if     iBD1020HFV_ma == iSMA                           // Simple moving average filter
static float32 BD1020HFV_sma_buf[iBD1020HFV_SMA_num];
static const sma_f32_t BD1020HFV_Phy_SMA =
{
        iInitial ,                                      // Initial state
        iBD1020HFV_SMA_num ,                          // Simple moving average number & buf size
        0U ,                                            // buffer position
        0.0F ,                                          // sum
        &BD1020HFV_sma_buf[0]                           // buffer
};
#elif   iBD1020HFV_ma == iEMA                           // Exponential moving average filter
static const ema_f32_t BD1020HFV_Phy_EMA =
{
        iInitial ,                                      // Initial state
        0.0F ,                                          // Xn-1
        iBD1020HFV_EMA_K                                // Exponential smoothing factor
};
#elif   iBD1020HFV_ma == iWMA                           // Weighted moving average filter
static float32 BD1020HFV_wma_buf[iBD1020HFV_WMA_num];
static const wma_f32_t BD1020HFV_Phy_WMA =
{
        iInitial ,                                      // Initial state
        iBD1020HFV_WMA_num ,                        // Weighted moving average number & buf size
        0U ,                                            // buffer poition
        iBD1020HFV_WMA_num * (iBD1020HFV_WMA_num + 1)/2 ,       // kn sum
        &BD1020HFV_wma_buf[0]                           // Xn buffer
};
#else                                                   // Non-moving average filter
#endif

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

```
const tbl_adc_t tbl_BD1020HFV =
{
        iBD1020HFV              ,
        iBD1020HFV_pin          ,
        iBD1020HFV_xoff         ,
        iBD1020HFV_yoff         ,
        iBD1020HFV_gain         ,
        iBD1020HFV_max          ,
        iBD1020HFV_min          ,
        iBD1020HFV_ma           ,

#if     iBD1020HFV_ma == iSMA                       // Simple moving average filter
        &BD1020HFV_Phy_SMA      ,
        (ema_f32_t*)iDummy_adr ,
        (wma_f32_t*)iDummy_adr
#elif   iBD1020HFV_ma == iEMA                       // Exponential moving average filter
        (sma_f32_t*)iDummy_adr ,
        &BD1020HFV_Phy_EMA      ,
        (wma_f32_t*)iDummy_adr
#elif   iBD1020HFV_ma == iWMA                       // Weighted moving average filter
        (sma_f32_t*)iDummy_adr ,
        (ema_f32_t*)iDummy_adr ,
        &BD1020HFV_Phy_WMA
#else                                               // Non-moving average filter
        (sma_f32_t*)iDummy_adr ,
        (ema_f32_t*)iDummy_adr ,
        (wma_f32_t*)iDummy_adr
#endif

};
```