# Specification document of OSS-EC BSL00000058

| | | | |
|---|---|---|---|
| BSL No. | 00000058 | | |
| Specification Ver | 01.00.00 | Sep 15,2022 | New release |
| | 01.00.01 | Sep 28,2022 | Corrected WMA |
| | 01.00.02 | Oct 19,2022 | Corrected license content |
| Documentation provided | Rui Long Lab Inc.  https://rui-long-lab.com/ | | |

License

1. Overview

This software specification is for OSS-EC for ADC components.

本ソフトウェア仕様は、ADC コンポーネント用 OSS-EC の仕様です。


2. Features

- Component type                    ADC component
- Number of Components          Single
- Supported OS (HAL)             Mbed
- Calculation                          Floating-point
- Conversion type                   Linear
- Moving average filter            Moving average filter select ( Non,SMA,EMA,WMA )
- Diagnosis                           Range (Min to Max)


3. OSS-EC Architecture

The OSS-EC architecture of ADC component (analog output component) is shown in the figure below, according to the component architecture.

ADC コンポーネント（アナログ出力コンポーネント）用 OSS-EC アーキテクチャは、コンポーネントアーキテクチャに合わせ、下図の通りです。

**fig. ADC component architecture & OSS-EC architecture**



ABCDEF : Model number

4. Components Software Interface

The sensor output voltage to physical quantity conversion is a linear conversion as in the following equation. The nonlinear conversion is provided in a separate BSL.

センサ出力電圧から物理量変換は、下式のような線形変換とする。尚、非線形変換は、別 BSL にて提供する。

ADC value to voltage value conversion formula

$$vi = ( ai \times ADC\_vdd ) / 2^{ADC\_bit}$$

Voltage value to physical value conversion formula

$$y = ( vi - x\_offset ) / gain + y\_offset \qquad \text{range min to max}$$

| | |
|---|---|
| y | physical value |
| ai | A/D conversion value |
| vi | Sensor output voltage value |
| ADC_vdd | Sensor supply voltage(Vdd) value |
| ADC_bit | A/D conversion bit length |
| x_offset | Sensor output voltage offset value |
| gain | Voltage value to physical value conversion gain |
| y_offset | Physical offset value |
| min | Physical value min |
| max | Physical value max |

5. Usage Scenes

The usage scenario of the ADC component is assumed to be a scene to be measured in cycles or events.
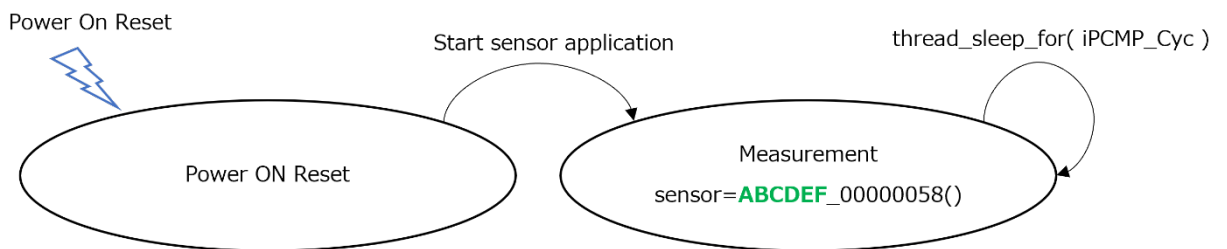
ADC コンポーネントの利用シーンは、周期またはイベントでの計測するシーンを想定する。

6. State flow

The state flow of the ADC component shall be as shown in the figure below from the usage scenes.

ADC コンポーネントの状態遷移は、利用シーンから下図の通りです。
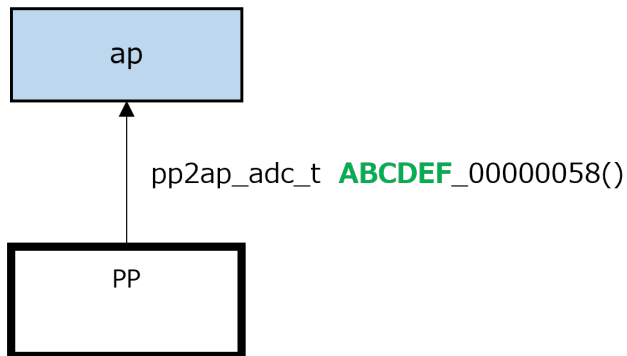
**fig. ADC component state flow**

7. API

The API is shown in the figure below.

API は、下図の通りです。

**fig 3. API**



```
typedef struct
{
        uint32                  sts;            // Diagnosis result of range min to max
        float32                 phy;            // Sensor physical value
}pp2ap_adc_t;
```

8. Outline specifications

The outline specifications of BSL00000058 are as follows

BSL00000058 の概略仕様は、下記の通りです。

1) Analog value read

2) Converts analog values to voltage values

3) Convert voltage values to physical values

4) Range min to max

| wk | range min to max | |
|---|---|---|
| res.sts | Normal | iNormal |
| | More than maximum value | iMax_NG |
| | Below minimum value | iMin_NG |

5) Moving average filter select

Select a moving average according to the value of iMA.

#define iMA    iSMA          Simple moving average filter

#define iMA    iEMA          Exponential moving average filter

#define iMA    iWMA          Weighted moving average filter

#define iMA    iNonMA        Non-moving average  filter

SMA calculation method

$$phy = ( y_n + y_{n-1} + y_{n-2} ) / n$$

n : iABCDEF_SMA_num

EMA calculation method

$$phy = ( y \times k ) + ( phy_{n-1} \times ( 1 - k ) )$$

k : iABCDEF_EMA_K

WMA calculation method

$$phy = ( ( y_n \times n ) + ( y_{n-1} \times (n-1) ) + \cdots + ( y_1 \times 1 ) / ( n + (n-1) + \cdots + 1 )$$

n : iABCDEF_WMA_num

```
#include "user_define.h"
#include "oss_ec_00000058.h"
static uint32 lib_f32_MaxMin( float32* , float32 , float32 );
#if     iMA == iSMA                             // Simple moving average filter
static float32 lib_f32_SMA( float32 , sma_f32_t* );
#elif   iMA == iEMA                             // Exponential moving average filter
static float32 lib_f32_EMA( float32 , ema_f32_t* );
#elif   iMA == iWMA                             // Weighted moving average filter
static float32 lib_f32_WMA( float32 , wma_f32_t* );
#else                                           // Non-moving average  filter
#endif

// ADC Main Function
pp2ap_adc_t pp_00000058( AnalogIn adc , tbl_adc_t tbl )
{
        pp2ap_adc_t res;

        // MP A/D value read
        uint16  ai = adc.read_u16();                            1)
        // CP A/D value to Voltage value conversion
        float32 vi = ( ai * iADC_vdd ) / ( 1<<iADC_bit );       2)
        // SP Voltage value to Physical value : linear conversion
        float32 wk = (( vi - tbl.xoff ) / tbl.gain ) + tbl.yoff; 3)
        // range (Min to Max)
        res.sts = lib_f32_MaxMin( &wk , tbl.max , tbl.min );    4)
        // Moving average filter select                         5)
#if     iMA == iSMA    // Simple moving average filter
        res.phy = lib_f32_SMA( wk , tbl.sma );
#elif   iMA == iEMA    //  Exponential moving average filter
        res.phy = lib_f32_EMA( wk , tbl.ema );
#elif   iMA == iWMA    // Weighted moving average filter
        res.phy = lib_f32_WMA( wk , tbl.wma );
#else                  // Non-moving average filter
        res.phy = wk;
#endif
        return( res );
}
```

9. Example

An example using XYZ's ABCDEF is shown below.

XYZ 社製 ABCDEF を用いた例は、下記の通りです。

```
void ap( void )
{
        pp2ap_adc_t   sensor;
        float         pressure;       // Pressure [kPa]
        unsigned long diagnosis;      // Diagnosis result
                                      //   Normal=iNormal,Max NG=iMax_NG,Min NG=iMin_NG
        // ABCDEF sensor read
        do{
                // Read of Pressure Sensor
                sensor = ABCDEF_00000058();
                pressure  = sensor.phy;
                diagnosis = sensor.sts;
                // Sensor Application
                // Application cycle wait
                thread_sleep_for( iPCMP_Cyc );
        }while(true);
}
```

10. File configuration

The file configuration of this OSS-EC is shown in the table below.

| Folder name | File name | Summary |
|---|---|---|
| XYZ_ABCDEF_00000058 | ABCDEF.cpp | ABCDEF const data file |
| | ABCDEF.h | ABCDEF define header file |
| | ABCDEF_00000058.cpp | ABCDEF code file |
| | user_setting.h | User setting header file |
| | pp_00000058.cpp | BSL00000058 code file |
| | oss_ec_00000058.h | BSL00000058 OSS-EC header file |
| | License.txt | OSS-EC license text file |
| XYZ_ABCDEF_00000058/documents | Spec-00000058.pdf | This specification pdf file |
| | Spec-ABCDEF.pdf | ABCDEF component software specification pdf file |
| | TestSpec&Result-ABCDEF.pdf | ABCDEF component test specifications and results |
| XYZ_ABCDEF_00000058/sample | main.cpp | Sample code file |

## 11. User setting

User configuration is done by user_setting.h. Select the moving average method, remove the comments as shown below. Also, change the coefficient of each moving average to match the product.

```
#define iADC_bit          16U          // MPU ADC bit
                                       // CAUTION : When Mbed is selected, 16 bit is fixed
#define iADC_vdd          5.0F         // MPU Vdd Configures the reference voltage [V]
#define iVref             5.0F         // Configures the reference voltage
#define iABCDEF_pin       A0           // ADC pin of ABCDEF


Case : Non-moving average filter select

#define iMA               iNonMA       // Non-moving average filter
//#define iMA              iSMA         // Simple moving average filter
//#define iMA              iEMA         // Exponential moving average filter
//#define iMA              iWMA         // Weighted moving average filter
//#define iABCDEF_SMA_num   4U          // Simple moving average number & buf size
//#define iABCDEF_EMA_K     0.25F       // Exponential Smoothing Factor
//#define iABCDEF_WMA_num   4U          // Weighted moving average number & buf size


Case : Simple moving average filter

//#define iMA              iNonMA       // Non-moving average filter
#define iMA               iSMA         // Simple moving average filter
//#define iMA              iEMA         // Exponential moving average filter
//#define iMA              iWMA         // Weighted moving average filter
#define iABCDEF_SMA_num   4U           // Simple moving average number & buf size
        CAUTION : iABCDEF_SMA_num > 0
//#define iABCDEF_EMA_K    0.25F        // Exponential Smoothing Factor
//#define iABCDEF_WMA_num   4U          // Weighted moving average number & buf size


Case : Exponential moving average filter

//#define iMA              iNonMA       // Non-moving average filter
//#define iMA              iSMA         // Simple moving average filter
#define iMA               iEMA         // Exponential moving average filter
//#define iMA              iWMA         // Weighted moving average filter
//#define iABCDEF_SMA_num   4U          // Simple moving average number & buf size
#define iABCDEF_EMA_K     0.25F        // Exponential Smoothing Factor
        CAUTION : 0.0 < iABCDEF_EMA_K < 1.0
//#define iABCDEF_WMA_num   4U          // Weighted moving average number & buf size


Case : Weighted moving average filter

//#define iMA              iNonMA       // Non-moving average filter
//#define iMA              iSMA         // Simple moving average filter
//#define iMA              iEMA         // Exponential moving average filter
#define iMA               iWMA         // Weighted moving average filter
//#define iABCDEF_SMA_num   4U          // Simple moving average number & buf size
//#define iABCDEF_EMA_K    0.25F        // Exponential Smoothing Factor
#define iABCDEF_WMA_num   4U           // Weighted moving average number & buf size
        CAUTION : iABCDEF_WMA_num > 0


#define iABCDEF_ma        iMA
```