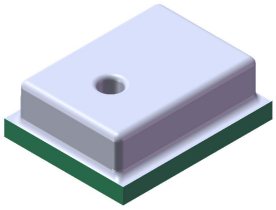
XGZP6816D BAROMETRIC PRESSURE SENSOR

FEATURES



- Wide Ranges: 300 ... 1100hPa
- 1.8V ~ 3.6V Power Supply
- Absolute Pressure Type
- Current Consumption:<80uA(single measurement at 128 OSR)
- Standby Current:<100nA (25°C)
- Calibrated Digital Signal(I2C Interface)
- Absolute Pressure Accuracy:±1hPa(8.3m)
- Relative Pressure Accuracy:±0.12hPa(1m)
- Temperature Accuracy:±2°C

APPLICATIONS

- Enhancement of GPS navigation (dead-reckoning, slope detection, etc.)
- In- and out-door navigation
- Leisure and sports
- Weather forecast
- Vertical velocity indication (rise/sink speed)

INTRODUCTION

XGZP6816D is a perfect silicon pressure sensor offering a ratiometric I2C interface for reading pressure over the specified full scale pressure span.

The XGZP6816D is a miniaturized Digital Barometric Air Pressure Sensor with a high accuracy and a low current consumption. The XGZP6816D is both a pressure and a temperature sensor. The pressure sensor element is based on a piezoresistive sensing principle which guarantees a high precision during temperature changes. The small package makes the XGZP6816D ideal for mobile applications and wearable devices.

The XGZP6816D's internal signal processor converts the output from the pressure and temperature sensor elements to 24-bit results. Each pressure sensor has been calibrated individually and contains calibration coefficients. The coefficients are used in the application to convert the measurement results to true pressure and temperature values.

XGZP6816D pressure sensor is for high volume application at an affordable cost but perfect performance.

PERFORMANCE PARAMETER

Unless otherwise specified, measurements were taken with a a temperature of 25±1℃ and humidity from 25% ~ 85%RH.

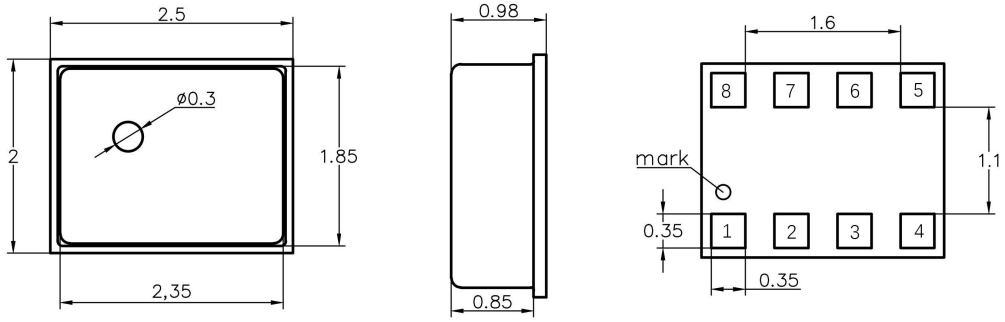
Parameter	Conditions	Min	Typ	Max	Units
Temp.Measure Range	Interior temp.sensor	-40		150	℃
Absolute Pressure Accuracy		-1		1	hPa
Relative Pressure Accuracy		-0.12		0.12	hPa
Temp. Measure Accuracy		-2		2	℃
Over Pressure			2x		Rated
Working Temp.		-40		85	℃
Compensated Temp.		0		60	℃

ELECTRICAL CHARACTERISTICS

Parameter	Conditions		Min	Typ	Max	Units
Average current during 1Hz conversion rate measurement	OSR_P	Oversampling rate 128x		80		μA
		Oversampling rate 64x		42		
		Oversampling rate 32x		23		
		Oversampling rate 16x		13		
		Oversampling rate 8x		8		
		Oversampling rate 3x		6		
		Oversampling rate 2x		4		
Peak Current				0.3		mA
Standby Current	Standby current in sleep state at 25°C			50	250	nA
Single measurement time (Including external bridge and temperature measurement time, the OSR of temperature measurement is 1024x)	OSR_P	Oversampling rate 128x		203		ms
		Oversampling rate 64x		105		
		Oversampling rate 32x		56		
		Oversampling rate 16x		31		
		Oversampling rate 8x		19		
		Oversampling rate 4x		13		
		Oversampling rate 2x		10		
ADC Conversion rate	OSR as 2x ~ 128x		20		1350	Hz
I2C Clock frequency					3.4	MHz
Temperature resolution				0.003		K/LSB

Parameter	Conditions	Min	Typ	Max	Units
Start Time	VDD to the time when the interface communication starts			1	ms
	VDD to the time when the measurement starts			2.5	ms
Wake up Time	Sleep status to the time when the interface communication starts			0.5	ms
	Sleep status to the time when the measurement starts			2	2

DIMENSION (Unit:mm Unspecified Tolerances:±0.1mm)

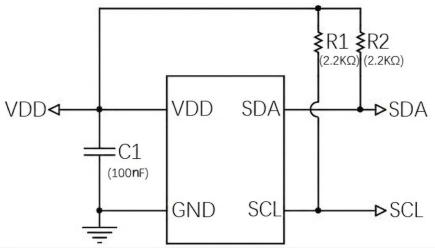


Front View Side View Bottom View

PIN DEFINITION

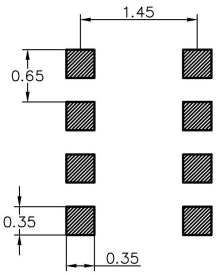
PIN 1/7	PIN3	PIN4	PIN8	PIN2/5/6
GND	SDA	SCL	VDD	N/C

SCL	The clock signal
NC	Do not connect to external circuitry or ground
GND	Ground
VDD	Voltage supply
SDA	Data signal(Send& Receive)



Recommended Application Circuit

FOOTPRINT(Unit:mm)

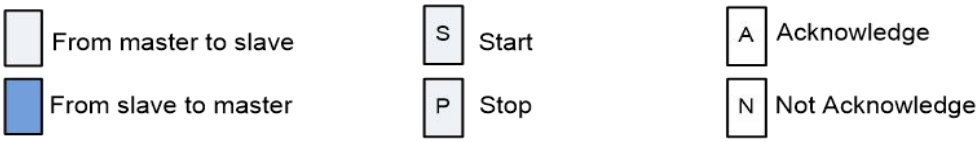


Symbol FootPrint 3D

NOTE: FOOTPRINT LAYOUT FOR REFERENCE ONLY CONTACT CFSENSOR FOR ABOVE FILE IF REQUIRED

I2C INTERFACE

The I2C bus uses SCL and SDA as signal lines, both of which are connected to VDD through pull-up resistors (typically 2.2K) and remain high level when not communicating.
The sensor is calibrated at the factory. Sending the 0xAC command to get the calibrated data..



Write Command

0xF0 means that the default 7bits I2C sensor slave device address is 0x78, and the last 1bit is 0 means that the master device MCU writes to the slave device. 0xAC is the command word to start the slave device sensor to perform a measurement. (The write address is 0X78<<1+0=0XF0, and the read address is 0X78<<1+1=0XF1)

Read Command

After sending the write command, need to wait for a while till measurement finish from the slave device sensor, and then send the read command to read the measurement data. Then sending the 0XF1 command to determine whether the sensor data acquisition has been completed

The waiting time depends on the settings of [13:11] Pressure Oversampling Rate of OTP (Address: 0x14) and [15:14] Temperature Oversampling Rate of OTP (Address: 0x14). The waiting time is =Tp+Tt.

Read AD Value

The read calibration data consists of 6 bytes, which are 1-byte status word, 3-byte pressure calibration value, and 2-byte temperature calibration value.

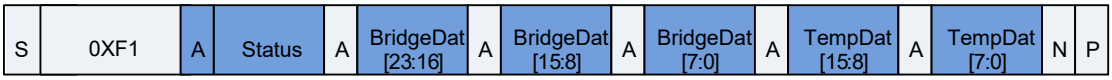


Table: Status of Bits

Bit	Significance	Description
Bit7	Reserved	Fixed Value: 0
Bit6	Power indication	1: Power on; 0: Power off
Bit5	Busy indication	1: Busy, the device is measuring pressure and temperature and the results are not ready yet. New I2C command will not be proceeded;

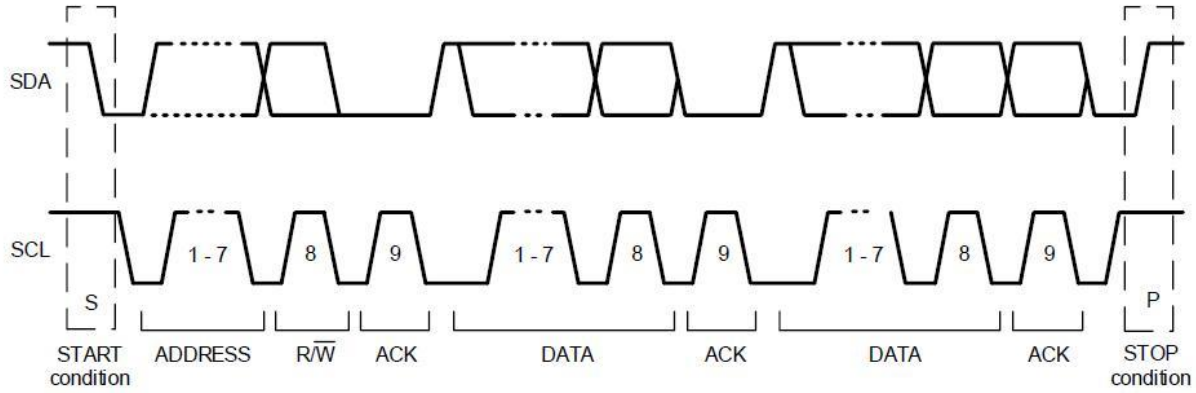
		0: Idle, the recent I2C command has been executed and the data to be read is ready
Bit4	Reserved	Fixed Value: 0
Bit3	Reserved	Fixed Value: 0
Bit2	Reserved	Fixed Value: 1
Bit1	Reserved	Fixed Value: 0
Bit0	Reserved	Fixed Value: 0

Sleep Standby

After the sensor completes the measurement, it enters the sleep standby state, and the standby power consumption is 0.1uA.

I2C TIME DIAGRAM

The I2C interface protocol has special bus signal conditions. Start (S), stop (P) and binary data conditions are shown below. At start condition, SCL is high and SDA has a falling edge. Then the slave address is sent. After the 7 address bits, the direction control bit R/W selects the read or write operation. When a slave device recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle.
At stop condition, SCL is also high, but SDA has a rising edge. Data must be held stable at SDA when SCL is high. Data can change value at SDA only when SCL is low.



I2C Command

Command(byte)	Return	Description	NOR Mode	CMD Mode
0x00~0x1F	16-bit data	Read data in the OTP that address matching command	Support	Support
0x40~0x5F Followed command byte: 0x0000 ~0xFFFF	—	Write data to OTP; Address is Command value subtract 0x40 (Address is 0x00 to 0x1F)	Support	Support
0xA0~0xA7 Followed command byte: 0xXXXX	24-bit raw data Get_Raw	Get_Raw Conduct one measurement, and write the raw ADC data to registers. See table 6-3 for further interpretation	Support	Support

0xA8	24-bit raw data Get_Raw	Start_NOM Quit CMD mode, enter NOR mode	No-Support	Support
0xA9	—	Start_CM Quit NOR mode, enter CMD mode	Support	No-Support
0xAA	—	Write_ChecksumC If CRC values are not wrote to OTP, the command check data in OTP register and writes CRC values to OTP	Support	Support
0xAC	24-bit compensated bridge data and 16-bit compensated temperature data	Get_Cal Measure based on OTP settings(AZBM, BM,AZTM and TM), write conpensated bridge and temperature data to I2C interface	Support	Support
0xB0~0xBF	24-bit compensated bridge data and 16-bit compensated temperature data	Get_Cal_S and Get_Cal are the same except that Get_Cal measures based on OTP defined OSR and Get_Cal_S measures based on command defined OSR, see following table	Support	Support

Get_Cal_S Command

Command 0xBX(HEX)	Function	Detail
X [3] Bit	OSR_T, ADC OSR of temperature measurement	0: 4x OSR 1: 8x OSR
X [2:0] Bit	OSR_P, ADC OSR of pressure measurement	000: 128x OSR 100: 8x OSR 001: 64x OSR 101:4x OSR 010: 32x OSR 110: 2x OSR 011: 16x OSR 000: 1x OSR

For example, to set the temperature ADC to 4x oversampling and the piezoelectric panel ADC to 1x oversampling, the command format is 0xB7,Just replace 0xAC with 0xB7

- Conversion Example:

After reading the calibration data, need to convert simply the unsigned number in the form of AD value..

To facilitate understanding, assuming that the calibration data read is: 0x04 0x9B 0xB0 0xC5 0x56 0xAA

0x04 is the status, and Bit5 is 1 indicating that the I2C is busy last time, and it needs to wait for a period of time. If Bit5 is 0, it indicates that the device is not busy and data can be read. For a detailed description of each bit of the status word, see above table"Status of Bits"

The three bytes of 0x9B, 0xB0, and 0xC5 are calibrated pressure value and the two bytes 0x56,0xAA are calibrated temperature value

The pressure calibration value is converted as follows:

Assuming that the range used in this calculation and calibration is 30Kpa as Pmin to 110Kpa as Pmax, the corresponding AD output is 1677722~15099494 (10%AD as Dmin_P~90%AD as Dmax_P).

Then convert 0x9B, 0xB0, and 0xC5 into decimal numbers, assuming it is 10203333(Dtest_P). According to the formula: Pressure= ((Pmax-Pmin)/(Dmax_P-Dmin_P)*(Dtest_P-Dmin_P)+Pmin), so,

Actual pressure value = (110-30)/(15099494-1677722)*(10203333-1677722)+30= 80.816(Unit: KPa)

The temperature calibration value is converted as follows:

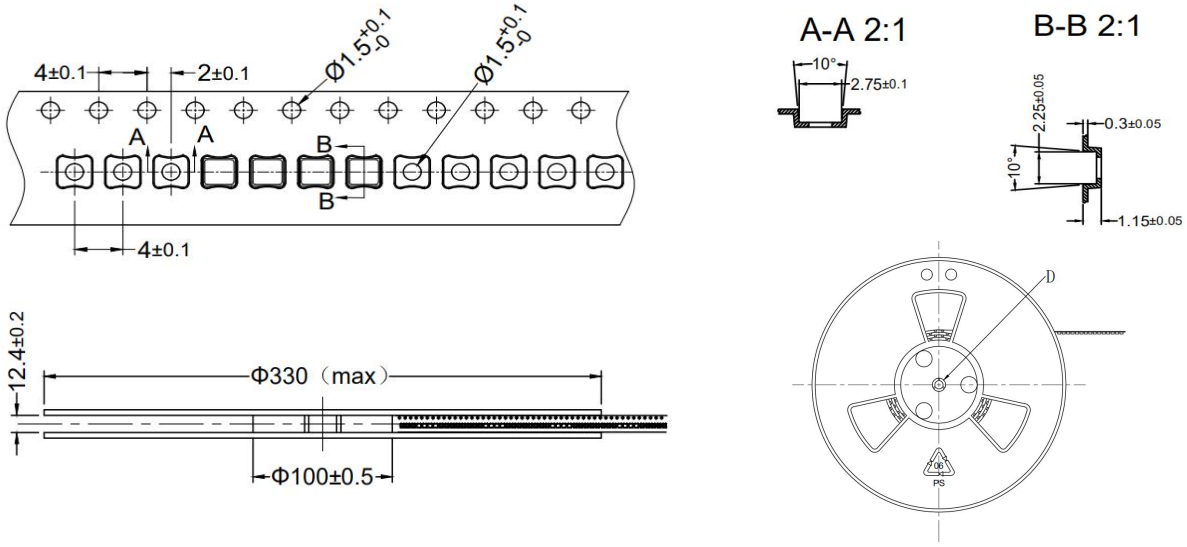
Assuming that the range used in this calculation and calibration is -40℃ as Tmin to 150℃ as Tmax, the corresponding AD output is 0~65535(0%AD as Dmin_T~100%AD as Dmax_T).

Then convert 0x56, 0xAA into decimal numbers, assuming it is 22186(Dtest_T). According to the formula: Temperature= ((Tmax-Tmin)/(Dmax_T-Dmin_T)*(Dtest_T-Dmin_T)+Tmin), so,

Actual temperature value = (150-(-40))/(65535-0)*(22186-0)+(-40)= 24.32(Unit:℃)

PACKING INFORMATION

Tape&Reel(unit: mm)
QTY/reel: 10,000 pcs.



OVERALL NOTES

Unless otherwise specified, following notes are general attention or presentation for all products from CFSensor.

Mounting

The following steps is for transmitting the air pressure to sensor after sensor soldering on PCB.

▼ For some sensors that come with inlet tube, select the flexible pipe to suit the pressure inlet that is firm enough to prevent the pressure leaks.

▼ Atmosphere hole (for Gauge type sensors) and Inlet pipe/hole can't be blocked with gel or glue etc,..

▼ Avoiding excessive external force operation

Soldering

Due to its small size, the thermal capacity of the pressure sensor is low. Therefore, take steps to minimize the effects of external heat. Damage and changes to characteristics may occur due to heat deformation. Use a non-corrosive resin type of flux. Since the pressure sensor is exposed to the atmosphere, do not allow flux to enter inside.

▼ Manual soldering

⊙Raise the temperature of the soldering tip between 260 and 300°C/500 and 572°F (30 W) and solder within 5 seconds.

⊙The sensor output may vary if the load is applied on the terminal during soldering.

⊙Keep the soldering tip clean.

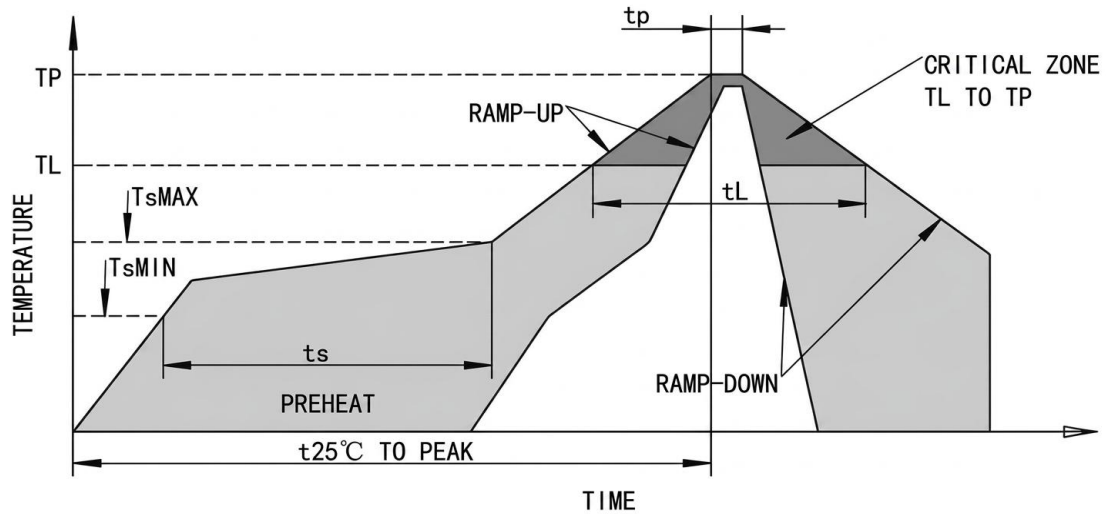
▼DIP soldering (DIP Terminal)

⊙Keep the temperature of the DIP solder tank below 260°C/500 and solder within 5 seconds.

⊙To avoid heat deformation, do not perform DIP soldering when mounting on the PCB which has a small thermal capacity.

▼Reflow soldering (SMD Terminal)

⊙The recommended reflow temperature profile conditions are given below.



Profile Feature	Pb-Free Assembly
Average ramp-up rate(TsMAX to TP)	3℃/seconds max.
Preheat -Temperature Min.(TsMIN) -Temperature Max.(TsMAX) -Time(TsMIN to TsMAX)(Ts)	 150℃ 200℃ 60 ~ 80seconds

Time maintained above: -Temperature(TL) -Time(tL)	 217℃ 60 ~ 150seconds
Peak temperature(TP)	260℃
Time within 5℃ of actual peak temperature(TP)2	20 ~ 40seconds
Ramp-down rate	4℃/seconds max.
Time 25℃ to peak temperature	8 minutes max.

⊙Self alignment may not always work as expected, therefore, please carefully note the position of the terminals and pattern.
⊙The temperature of the profile is assumed to be a value measured with the PCB of the terminal neighborhood.
⊙Please evaluate solderability under the actual mounting conditions since welding and deformation of the pressure inlet port may occur due to heat stress depending on equipments or conditions.

▼ Rework soldering
⊙Complete rework at a time.
⊙Use a flattened soldering tip when performing rework on the solder bridge. Do not add the flux.
⊙Keep the soldering tip below the temperature described in the specifications.

▼ Avoid drop and rough handling as excessive force may deform the terminal and damage soldering characteristics.
▼ Keep the circuit board warpage within 0.05 mm of the full width of the sensor.
▼ After soldering, do not apply stress on the soldered part when cutting or bending the circuit board.
▼ Prevent human hands or metal pieces from contacting with the sensor terminal. Such contact may cause anomalous outlets as the terminal is exposed to the atmosphere.
▼ After soldering, prevent chemical agents from adhering to the sensor when applying coating to avoid insulation deterioration of the circuit board.

Connecting

▼ Correctly wire as the connection diagram. Reverse connection may damage the product and degrade the performance.
▼ Do not use idle terminals(N/C) to prevent damages to the sensor.

Cleaning

▼ Since the pressure sensor is exposed to the atmosphere, do not allow cleaning fluid to enter inside from atmosphere hole (for Gauge type sensors) and inlet pipe.
▼ Avoid ultrasonic cleaning since this may cause breaks or disconnections in the wiring.

Environment

▼Please avoid using or storing the pressure sensor in a place exposed to corrosive gases (such as the gases given off by organic solvents, sulfurous acid gas, hydrogen sulfides, etc.) which will adversely affect the performance of the pressure sensor chip.
▼Since this pressure sensor itself does not have a water-proof construction(even available media can be liquid), please do not use the sensor in a location where it may be sprayed with water, etc.
▼Avoid using the pressure sensors in an environment where condensation may form. Furthermore, its output may fluctuate if any moisture adhering to it freezes.
▼The pressure sensor is constructed in such a way that its output will fluctuate when it is exposed to light. Especially when pressure is to be applied by means of a transparent tube, take steps to prevent the pressure sensor chip from being exposed to light.

▼Avoid using pressure sensor where it will be susceptible to ultrasonic or other high-frequency vibration.

▼Keeping the sensors sealed in static shielding bags with an oxygen-free condition and use the sensor as soon as possible once unfold the package, because the sensors' PINs may be oxidated a bit under atmosphere environment(slight oxidation wouldn't affect soldering and performance)

More Precautions

▼That using the wrong pressure range or mounting method may result in accidents.

▼The only direct pressure medium you can use is non-corrosive gas or air as illuminated above(Note: some sensors are compatible with liquid media). The use of other media, in particular, corrosive gases and liquid (organic solvent based, sulfurous acid based, and hydrogen sulfide based, etc.) or contains foreign substances will cause malfunction and damage. Please do not use them and check with CFSensor.

▼The pressure sensor is positioned inside the pressure inlet. Never poke wires or other foreign matter through the pressure inlet since they may damage the sensor or block the inlet. Avoid use when the atmospheric pressure inlet(only for Gauge type pressure sensor) is blocked.

▼Use an operating pressure which is within the rated pressure range. Using a pressure beyond this range may cause damage.

▼Since static charge can damage the pressure sensor, bear in mind the following handling precautions.

⊙When storing the pressure sensor, use a conductive material to short the pins or wrap the entire sensor in aluminum foil. Common plastic containers should not be used to store or transport the sensor since they readily become charged.

⊙When using the pressure sensor, all the charged articles on the bench surface and the work personnel should be grounded so that any ambient static will be safely discharged.

▼Based on the pressure involved, give due consideration to the securing of the pressure sensor.

【 SAFETY NOTES 】

Using these sensors products may malfunction due to external interference and surges, therefore, please confirm the performance and quality in actual use. Just in case, please make a safety design on the device (fuse, circuit breaker, such as the installation of protection circuits, multiple devices, etc.), so it would not harm life, body, property, etc even a malfunction occurs. To prevent injuries and accidents, please be sure to observe the following items:

- The driving current and voltage should be used below the rated value.
- Please follow the terminal connection diagram for wiring. Especially for the reverse connection of the power supply, it will cause an accident due to circuit damage such as heat, smoke, fire, etc.
- In order to ensure safety, especially for important uses, please be sure to consider double safety circuit configuration.
- Do not apply pressure above the maximum applied pressure. In addition, please be careful not to mix foreign matter into the pressure medium. Otherwise, the sensor will be discarded, or the media will blew out and cause an accident.
- Be careful when fixing the product and connecting the pressure inlet. Otherwise, accidents may occur due to sensor scattering and the blowing out of the media.
- If the sensor come with sharp PIN, please be careful not to hurt your body when using it.

【 WARRANTY 】

The information in this sheet has been carefully reviewed and is believed to be accurate; however, no responsibility is assumed for inaccuracies. Furthermore, this information does not convey to the purchaser of such devices any license under the patent rights to the manufacturer. CFSensor reserves the right to make changes without further notice to any product herein. CFSensor makes no warranty, representation or guarantee regarding the suitability of its product for any particular purpose, nor does CFSensor assume any liability arising out of the application or use of any product or circuit and specifically disclaims any and all liability, including without limitation consequential or incidental damages. Typical parameters can and do vary in different applications. All operating parameters must be validated for each customer application by customer's technical experts. CFSensor does not convey any license under its patent rights nor the rights of others.

【 CONTACT 】

CFSensor
22F/14Bldg High-Tech Park High-Tech Area Wuhu P.R.C.241000
Tel/Fax:+86 18226771331 Email:INFO@CFSensor.com

North America II Europe II Southeast Asia II Middle East II Latin America

```
/*****************************************************************************
 * File: XGZP6816D_Sensor_Driver.c
 * Description: I2C address 0x78 pressure/temperature sensor driver (STM32F103, Software I2C)
 * Hardware: SCL-PA6, SDA-PA7, UART-PA9(TX)/PA10(RX)
 *****************************************************************************/

#include "stm32f10x.h"
#include <stdio.h>
#include <math.h>

// ==================== Debug Switch ====================
#define DEBUG 1                    // 1: Enable debug print, 0: Disable

// ==================== Sensor Parameters ====================
#define ADC_FULL_SCALE      16777216UL      // 24-bit ADC full scale (2^24)
#define PMIN               30.0f          // Sensor minimum range (KPa)
#define PMAX               110.0f         // Sensor maximum range (KPa)
#define DMIN               (ADC_FULL_SCALE * 0.15f)   // AD value at minimum range (15%)
#define DMAX               (ADC_FULL_SCALE * 0.85f)   // AD value at maximum range (85%)
#define SENSOR_ADDRESS      0x78           // 7-bit I2C address of the sensor
#define CMD_MEASURE         0xAC           // Measurement command
#define SEA_LEVEL_PRESSURE  101325.0f       // Standard sea level pressure (Pa)
#define WAIT_TIMEOUT        200            // Wait timeout count

// ==================== I2C Pin Definition (PA6=SCL, PA7=SDA) ====================
#define I2C_GPIO           GPIOA
#define SCL_PIN            GPIO_Pin_6
#define SDA_PIN            GPIO_Pin_7
#define I2C_CLOCK          RCC_APB2Periph_GPIOA

#define SDA_HIGH()         I2C_GPIO->BSRR = SDA_PIN
#define SDA_LOW()          I2C_GPIO->BRR  = SDA_PIN
#define SCL_HIGH()         I2C_GPIO->BSRR = SCL_PIN
#define SCL_LOW()          I2C_GPIO->BRR  = SCL_PIN
#define READ_SDA()         (I2C_GPIO->IDR & SDA_PIN)

#define DELAY_TIME         2              // I2C delay (microseconds)
#define SUCCESS            0
#define FAILED             1

// ==================== Global Variables ====================
float pressure_kpa = 0.0;          // Pressure value (KPa)
```

```
double pressure_pa = 0.0;          // Pressure value (Pa)
float temperature = 0.0;           // Temperature value (℃)
float altitude = 0.0;              // Altitude (meters)

// ==================== Delay Functions ====================
static uint32_t fac_us = 0;

void Delay_Init(void)
{
    SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div8);
    fac_us = SystemCoreClock / 8000000;
}

void Delay_Us(uint32_t nus)
{
    uint32_t temp;
    SysTick->LOAD = nus * fac_us;
    SysTick->VAL = 0x00;
    SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;
    do {
        temp = SysTick->CTRL;
    } while ((temp & 0x01) && !(temp & (1 << 16)));
    SysTick->CTRL &= ~SysTick_CTRL_ENABLE_Msk;
    SysTick->VAL = 0x00;
}

void Delay_Ms(uint32_t nms)
{
    uint32_t i;
    for (i = 0; i < nms; i++) {
        Delay_Us(1000);
    }
}

// ==================== UART Initialization (for printf) ====================
void UART_Init(uint32_t baudrate)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 | RCC_APB2Periph_GPIOA, ENABLE);
```

```
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    USART_InitStructure.USART_BaudRate = baudrate;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART1, &USART_InitStructure);

    USART_Cmd(USART1, ENABLE);
}

int fputc(int ch, FILE *f)
{
    USART_SendData(USART1, (uint8_t)ch);
    while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
    return ch;
}

// ==================== I2C Low-Level Functions ====================
void IIC_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(I2C_CLOCK, ENABLE);

    GPIO_InitStructure.GPIO_Pin = SCL_PIN | SDA_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_OD;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(I2C_GPIO, &GPIO_InitStructure);

    GPIO_SetBits(I2C_GPIO, SCL_PIN | SDA_PIN);
}

uint8_t IIC_Start(void)
```

```
{
    SDA_HIGH();
    SCL_HIGH();
    Delay_Us(DELAY_TIME);

    if (!READ_SDA())
        return FAILED;

    SDA_LOW();
    Delay_Us(DELAY_TIME);

    SCL_LOW();
    Delay_Us(DELAY_TIME);

    return SUCCESS;
}

void IIC_Stop(void)
{
    SDA_LOW();
    SCL_LOW();
    Delay_Us(DELAY_TIME);

    SCL_HIGH();
    Delay_Us(DELAY_TIME);

    SDA_HIGH();
    Delay_Us(DELAY_TIME);
}

void IIC_Ack(void)
{
    SDA_LOW();
    Delay_Us(DELAY_TIME);

    SCL_HIGH();
    Delay_Us(DELAY_TIME);

    SCL_LOW();
    Delay_Us(DELAY_TIME);

    SDA_HIGH();
```

```
}

void IIC_NAck(void)
{
    SDA_HIGH();
    Delay_Us(DELAY_TIME);

    SCL_HIGH();
    Delay_Us(DELAY_TIME);

    Delay_Us(DELAY_TIME);

    SCL_LOW();
    Delay_Us(DELAY_TIME);

    SDA_HIGH();
}

uint8_t IIC_Wait_Ack(void)
{
    SDA_HIGH();
    SCL_HIGH();
    Delay_Us(DELAY_TIME);

    if (READ_SDA()) {
        SCL_LOW();
        return FAILED;
    }

    SCL_LOW();
    Delay_Us(DELAY_TIME);
    return SUCCESS;
}

void IIC_Send_Byte(uint8_t byte)
{
    uint8_t i = 8;
    while (i--) {
        SCL_LOW();
        Delay_Us(DELAY_TIME);

        if (byte & 0x80)
```

```
            SDA_HIGH();
        else
            SDA_LOW();

        byte <<= 1;
        Delay_Us(DELAY_TIME);

        SCL_HIGH();
        Delay_Us(DELAY_TIME);
    }
    SCL_LOW();
    Delay_Us(DELAY_TIME);
}

uint8_t IIC_Read_Byte(void)
{
    uint8_t i = 8;
    uint8_t byte = 0;

    SDA_HIGH();
    while (i--) {
        byte <<= 1;
        SCL_LOW();
        Delay_Us(DELAY_TIME);

        SCL_HIGH();
        Delay_Us(DELAY_TIME);

        if (READ_SDA()) {
            byte |= 0x01;
        }
    }
    SCL_LOW();
    Delay_Us(DELAY_TIME);
    return byte;
}

// ==================== I2C Read/Write Functions ====================
uint8_t I2C_WriteData(uint8_t address, uint8_t *buf, uint8_t count)
{
    IIC_Start();
    IIC_Send_Byte((address << 1) & 0xFE);
```

```
    if (IIC_Wait_Ack() == FAILED) {
        IIC_Stop();
        return FAILED;
    }

    while (count--) {
        IIC_Send_Byte(*buf++);
        if (IIC_Wait_Ack() == FAILED) {
            IIC_Stop();
            return FAILED;
        }
    }

    IIC_Stop();
    return SUCCESS;
}

uint8_t I2C_ReadData(uint8_t address, uint8_t *buf, uint8_t count)
{
    IIC_Start();
    IIC_Send_Byte((address << 1) | 0x01);

    if (IIC_Wait_Ack() == FAILED) {
        IIC_Stop();
        return FAILED;
    }

    while (count--) {
        *buf = IIC_Read_Byte();
        if (count > 0) {
            IIC_Ack();
        } else {
            IIC_NAck();
        }
        buf++;
    }

    IIC_Stop();
    return SUCCESS;
}
```

```
// ==================== Sensor Functions ====================
uint8_t Sensor_IsBusy(void)
{
    uint8_t status;
    I2C_ReadData(SENSOR_ADDRESS, &status, 1);
    status = (status >> 5) & 0x01;
    return status;
}

void Sensor_GetData(void)
{
    uint8_t buffer[6] = {0};
    uint32_t pressure_AD = 0;
    uint16_t temperature_AD = 0;
    uint8_t timeout = 0;

    // 1. Send measurement command
    buffer[0] = CMD_MEASURE;
    I2C_WriteData(SENSOR_ADDRESS, buffer, 1);

    // 2. Wait for measurement to complete
    while (Sensor_IsBusy()) {
        Delay_Ms(1);
        timeout++;
        if (timeout > WAIT_TIMEOUT) {
            #if DEBUG
            printf("Warning: Sensor measurement timeout!\r\n");
            #endif
            break;
        }
    }

    // 3. Read 6 bytes of data
    I2C_ReadData(SENSOR_ADDRESS, buffer, 6);

    // 4. Parse AD values
    pressure_AD = (uint32_t)(((uint32_t)buffer[1] << 16) |
                          ((uint32_t)buffer[2] << 8) |
                          (uint32_t)buffer[3]);
    temperature_AD = (uint16_t)(((uint16_t)buffer[4] << 8) | buffer[5]);

    // 5. Calculate pressure (KPa)
```

```
    if (pressure_AD != 0) {
        pressure_kpa = (float)((PMAX - PMIN) / (DMAX - DMIN) *
                               (pressure_AD - DMIN) + PMIN);
        pressure_pa = (double)(pressure_kpa * 1000.0);
    } else {
        pressure_kpa = 0.0f;
        pressure_pa = 0.0;
    }

    // 6. Calculate temperature (℃)
    temperature = ((float)temperature_AD / 65536.0f * 19000.0f - 4000.0f) / 100.0f;

    // 7. Calculate altitude (meters)
    altitude = 44330.0f * (1.0f - powf((float)(pressure_pa / SEA_LEVEL_PRESSURE), 1.0f / 5.255f));

    #if DEBUG
    printf("========== 0x78 Sensor Data ==========\r\n");
    printf("buffer: %02X %02X %02X %02X %02X %02X\r\n",
           buffer[0], buffer[1], buffer[2], buffer[3], buffer[4], buffer[5]);
    printf("pressure_AD = %lu\r\n", pressure_AD);
    printf("temperature_AD = %u\r\n", temperature_AD);
    printf("Pressure: %.3f Pa (%.3f KPa)\r\n", pressure_pa, pressure_kpa);
    printf("Temperature: %.2f ℃\r\n", temperature);
    printf("Altitude: %.2f m\r\n", altitude);
    printf("======================================\r\n\r\n");
    #endif
}

// ==================== Main Function ====================
int main(void)
{
    // System initialization
    Delay_Init();
    UART_Init(115200);
    IIC_Init();

    printf("\r\n=== 0x78 Sensor Driver ===\r\n");
    printf("System Ready!\r\n\r\n");

    while (1) {
        Sensor_GetData();
        Delay_Ms(1000);
```

```
    }
}
```