I2C Digital Output Read Process

This process is applicable for reading calibrated digital outputs.

—I2C device addresses are as follows:

表 5.4 I2C 通配地址

A7	A6	A5	A4	A3	A2	A1	W/R
1	1	1	1	1	1	1	0/1

The data reading process is carried out in the following order:

1. Send command 0x0A to register 0x30 for temperature acquisition and pressure data acquisition.

2. Read the value of register 0x30. If the Sco bit (bit 3) is 0, the acquisition is complete and data can be read. Alternatively, wait with a delay (e.g., 50ms).

3. Read data from registers 0x06, 0x07, 0x08, 0x09, and 0x0A.

Among these, registers REG0x06, REG0x07, and REG0x08 hold the 24-bit ADC values for pressure output. Among these, 0x06 contains the high 8 bits, and 0x08 contains the low 8 bits. The highest bit in the 24 bits is

the sign bit. When the sign bit value is "1," it represents "negative," and when the sign bit value is "0," it represents "positive."

Registers REG0x09 and REG0x0A contain the 16-bit temperature output values. Among these, 0x09 contains the high 8 bits, and the highest bit is the sign bit. When the sign bit value is "1," it represents "negative," and when the sign bit value is "0," it represents "positive."

For example, the decimal values read from REG0x06, REG0x07, REG0x08, REG0x09, and REG0x0A are denoted as x, y, z, a, and b.

Pressure ADC value: $m = x * 2^{16} + y * 2^8 + z$

Positive/Negative handling: If $m > 2^{23}$, the pressure signal is negative, and pressure value = $(m - 2^{24}) / 2^{23} *$ Fullscale;

If $m < 2^{23}$, the pressure signal is positive, and pressure value = $m / 2^{23} *$ Fullscale;

Here, Fullscale represents the sensor's full-scale pressure.

Temperature value: n = a * 2^8 + b

Positive/Negative handling: If n > 2^15, it's a negative value, and temperature value = (n - 2^16) / 256 (°C);

If n < 2^15, it's a positive value, and temperature value = n / 256 (°C);

The data handling routine is as follows: This routine reads temperature and pressure data sequentially and is provided for customer reference.

```c
#include <stdio.h>

int main(void)
{
    uint8_t Pressure[3];
    uint8_t Temp[2];
    char data[3];
    float  Cal_PData1;          // 24-bit AD value - pressure data
```

```
        float Cal_PData2;          // Current pressure as
a percentage of full-scale pressure
        float Pressure_data;       // Current actual
pressure output

        float Cal_TData1;          // 24-bit AD value -
temperature data
        float Cal_TData2;          // Current temperature
as a percentage of full-scale temperature
        float Temp_data;           // Current actual
temperature output

        float Fullscale_P;         // Full-scale pressure
value
        Fullscale_P = 1000;        // Please define the
full-scale pressure (e.g., 1000 kPa)

        while (1)
        {
            I2C_WriteReg(0x30, 0x0A);
            delay_ms(100);
            I2C_ReadNByte(0x06, Pressure, 3); //
```

```
Read registers 0x06, 0x07, and 0x08 sequentially.
                data[0] = Pressure[0];
                data[1] = Pressure[1];
                data[2] = Pressure[2];
                Cal_PData1 = data[0] * 65535 + data[1] *
256 + data[2];
                if (Cal_PData1 > 8388608)
                {
                      Cal_PData2     =     (Cal_PData1     -
16777216) / 8388608;
                }
                else
                {
                      Cal_PData2 = Cal_PData1 / 8388608;
                }
                Pressure_data = Cal_PData2 * Fullscale_P;
// Calculate pressure value

                I2C_ReadNByte(0x09,  Temp,  2);  //  Read
registers 0x09 and 0x0A sequentially.
                data[0] = Temp[0];
                data[1] = Temp[1];
```

```
                Cal_TData1 = data[0] * 256 + data[1];
                if (Cal_TData1 > 32768)
                {
                        Cal_TData2 = (Cal_TData1 - 65536) /
256;
                }
                else
                {
                        Cal_TData2 = Cal_TData1 / 256;
                }
                Temp_data = Cal_TData2; // Calculate
temperature value

                // Use Pressure_data and Temp_data as
needed in your application
            }

            return 0;
        }
        ```

        Note: `Pressure_data` represents the current actual
```
```

pressure output in the environment, and `Temp_data` represents the current actual temperature in the environment.

Wiring Configuration:

- Brown Wire: Power Supply (+)
- Blue Wire: Ground (GND)
- White Wire: Serial Data Line (SDA)
- Black Wire: Serial Clock Line (SCL)