1.0 General Description

1.1 Scope

This application note is limited to AVSensors digital series pressure sensors with 16bit resolution. The code example should be used in conjunction the MCT Digital Output 16 Bit application note.

1.2 AVSensors, Multi Chip Technology, Digital Series

One of AVSensors' key advantages is its use of multi-chip technology, which provides exceptional flexibility in design. By integrating multiple chips into a single sensor package, AVSensors delivers a broad range of digital pressure sensors that balance resolution, performance, and cost to meet the varied requirements of OEM applications.

The 14-bit model delivers 14-bit pressure and 11-bit temperature outputs, operates on either 3.3 V or 5.0 V, and features second-order correction for offset, sensitivity over temperature, and pressure nonlinearity—providing a strong combination of accuracy and affordability for cost-sensitive designs.

The 16-bit version offers full 16-bit resolution for both pressure and temperature with the same dual-voltage flexibility, making it ideal for applications that demand higher precision without a significant cost increase.

2.0 Code Examples

2.1 Introduction

In the realm of digital pressure sensing, efficient communication protocols are indispensable for seamless integration into various applications. The Inter-Integrated Circuit (I2C) and Serial Peripheral Interface (SPI) protocols stand out as two widely adopted standards, offering flexible and reliable means of interfacing with digital pressure sensors. However, despite their prevalence, navigating the intricacies of these protocols can pose challenges for developers, especially those new to the field. To facilitate the implementation process and empower developers with practical insights, this application note delves into the fundamentals of I2C and SPI interfacing for digital pressure sensors, accompanied by comprehensive code examples. Through elucidating these protocols and providing hands-on examples, this document aims to streamline the integration process, enabling developers to leverage.

2.2 I2C Code Example

```
/*****************************************************************************
Function Name: SKATER_I2C_MultRead
* Description: Writes or reads multiple bytes at a specified address (7-bit address).
* Input: - pBuffer: Data storage area.
* - I2C_SLAVE_ADDRESS: Device address.
* - RegAddr: Starting register address.
* - NumByteToRead: Number of bytes to read consecutively.
* Output: None
* Return: None
*****************************************************************************/
void I2C_MultiRead(uint8_t* data, uint8_t dev, uint8_t reg, uint8_t NumByteToRead)
{
uint8_t count=0;
IIC_Start();                                          //Generate a start condition
IIC_Send_Byte(dev);                                   //Send write command
IIC_Wait_Ack();                                       //Response
IIC_Send_Byte(reg);                                   //Send address
IIC_Wait_Ack();                                       //Response
IIC_Start();                                          //Generate a start condition
IIC_Send_Byte(dev+1);                                 //Enter receive mode
IIC_Wait_Ack();                                       //Response
for(count=0;count<NumByteToRead;count++){
if(count!=NumByteToRead-1)data[count]=IIC_Read_Byte(1);   //Read data with ACK
else data[count]=IIC_Read_Byte(0);                    //NACK the last byte
}
IIC_Stop();                                           //Generate a stop condition
return;
}
```


```
float GetPa(void)
{
uint8_t data[6]={0};
int16_t PaD=0,Temp=0;
float Per[6]={0},Psum=0,Pmax=-1000,Pmin=1000;          //Psum total, Pmax: maximum pressure
                                                       //value, Pmin: minimum pressure value
Pa_flag=0;
for(uint8_t i=0;i<4;i++)
{
// HAL_I2C_Mem_Read(&hi2c1,0xD8, 0x2E, I2C_MEMADD_SIZE_8BIT,data, 6, 1000);

I2C_MutiRead(data, 0xD8,0x2E,6);          //Read sensor data and put it into data, data: data //buff,
                                          0xD8: slave address, 0x2E: temperature register address, 6:
                                          read 6 consecutively;

Temp=(data[1]<<8)+data[0];                //Temperature: data[1]: data high 8 bits, data[0]:
                                          //data low 8 bits
PaD=(data[3]<<8)+data[2];                 //Pressure size, high address + low address
Per[i]=(-500.0+(PaD+26214)*1000.0/52428)*1.0;   //6895
if(Per[i]>Pmax)
Pmax=Per[i];
if(Per[i]<Pmin)
Pmin=Per[i];
Psum+=Per[i];
}
Pa=(Psum-Pmax-Pmin)/2-FlashParameter.Offset;
Tempr=-40-2+(Temp+32768)*165/65535;          //2 degree deviation
Pa_flag=1;
if(FlashParameter.Range==1)                  //40-50Pa
{
if(fabs(Pa)>=50)
Alarm=1;
else if(fabs(Pa)<=40)
Alarm=0;
}
else                                         //25-30Pa
{
if(fabs(Pa)>=30)
Alarm=1;
else if(fabs(Pa)<=25)
Alarm=0;
```


```
}
return Pa;
}
```