Key Features

- Piezo-resistive silicon micro-machined sensor
- Gauge type pressure sensor
- I^{2}C Interface
- Pressure range: 0 to +300 kPa
- Pressure Sensitivity: 0.0004 kPa/LSB
- 24 Bit Σ-Δ ADC (ADC Resolution Setting@20 Bit)
- Temperature Compensation: 0 ~ 50 ^{o}C
- Operating voltage 3.0V
- Operating mode current: ~0.6mA (typical)
- Sleep Mode current: 20nA (typical)
- SOP6 package
- RoHS compliant and Halogen-free

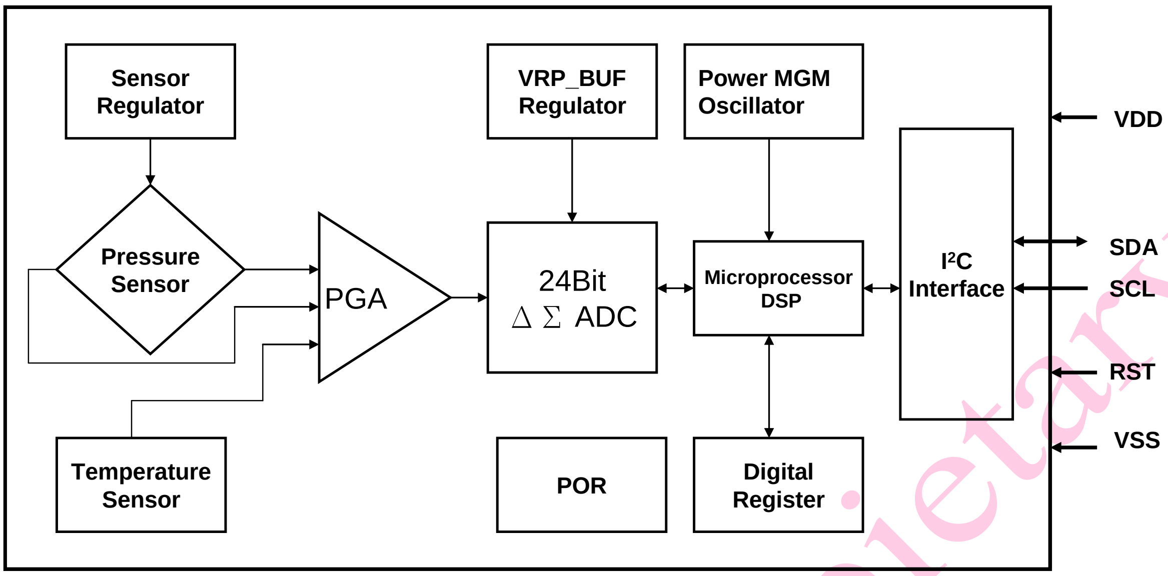
Applications

- Medical instrumentation
- Industrial pneumatic control
- Air Conditioning
- HVAC Application
- Home electricity appliances

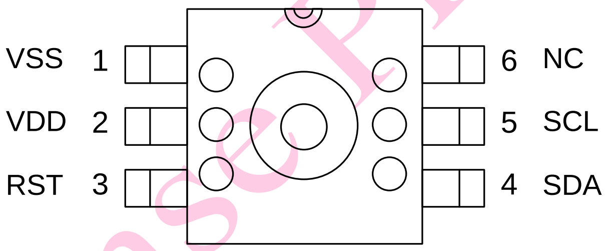
Description

The US6330 is the pressure sensor which measures gauge pressures. It consists of a silicon micro-machined sensing element chip and a signal conditioning ASIC. The ASIC is equipped with a 24-bit resolution Σ-Δ ADC and outputs a highly precise pressure value as a digital value. The pressure sensor element and the ASIC are mounted inside a system-in-package and wire-bonded to appropriate contacts. The US6330 provides digital output interface. It can achieve ESD robustness, fast response time, high accuracy and linearity as well as long-term stability. All measurement data is fully calibrated and temperature compensated. In addition, it allows for easy system integration.

Block Diagram



Pin Configuration



Top View

Pin Description

Pin No.	Pin Name	I/O	Function description
1	VSS	S	Connected to GND
2	VDD	S	Positive supply voltage
3	RST	I	Device Reset (Low active, internal is pull-up)
4	SDA	I/O	Data in/out
5	SCL	I	Clock input
6	NC	—	No connected

Note:

1*) Pin Type: S = Supply, I= Input, O = Output.

Maximum Ratings (Voltage with respect to GND unless otherwise noted)

VDD	- 0.4 V to +3.63 V
Voltage at Digital IO Pins	- 0.5 V to VDD+0.5 V
Operating Temperature Range	- 40°C to +85°C
Storage Temperature Range	- 40°C to +125°C
Electrostatic Discharge Tolerance – Human Body Model	± 2kV
Pressure Medium	Air (Note1)

Note1: This medium must be dry and non-corrosive.

Specification

Electrical Characteristics (VDD=3V unless otherwise noted.)

Power Supply Characteristics						
Symbol	**Parameter**	**Conditions**	**Min.**	**Typ.**	**Max.**	**Unit**
V_{DD}	Operating Voltage	—	1.68	3.0	3.6	V
V_{DDB}	Reference Voltage	Internal power source of the MEMS sensor	1.60	1.68	1.75	V
I_{DD}	Operating Current	VDD=3.0V	—	0.6	1.2	mA
I_{STB1}	Standby Current	VDD=3.0V , Temperature ≦ 85℃	—	20	250	nA
Pressure Output Characteristics						
Symbol	**Parameter**	**Conditions**	**Min.**	**Typ.**	**Max.**	**Unit**
P_{PRO}	Proof pressure	—	—	—	+3	MPa
P_{OP}	Operating pressure	—	0	—	+300	kPa
P_{RACC}	Relative Accuracy	P: 0 ~ +300 kPa , T: 25℃	—	±1.0	—	kPa
P_{AACC1}	Absolute Accuracy	P: 0 ~ +300 kPa , T: 0 ~ +50℃	-3.0	—	+3.0	kPa
P_{NS}	Resolution (RMS)	—	—	0.0004	—	kPa/LSB
P_{SEN}	Pressure Sensitivity	—	—	0.005	—	kPa
ADC Characteristics						
Symbol	**Parameter**	**Conditions**	**Min.**	**Typ.**	**Max.**	**Unit**
P_{RES}	Pressure Resolution	Factory Option	—	20	—	bit
t_{ADC}	AD Conversion Time	For a signal Analog-to-Digital conversion time	—	1.7	—	ms
t_{update}	Data Update Time	Full measurement and temperature compensation	—	6.6	—	ms
t_{ST}	Start-Up Time	VDD ramp up to communication	—	—	1	ms
		VDD ramp up to Analog operation	—	—	2.5	ms
t_{SLP}	Wake-UP Time	Sleep to Interface communication	—	—	0.5	ms
		Sleep to analog operation	—	—	2	ms

Interface/Frequency Characteristics

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Unit
Fsys	System Frequency	—	—	4	—	MHz
F$_{I2C}$	I^2C Clock Frequency	—	—	—	3.4	MHz

Temperature Output Characteristics (VDD=3V unless otherwise noted)

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Unit
Trg	Measurement Range	—	0	—	+50	℃
Tacc	Absolute Accuracy	0 ~ 50 ℃	—	± 3	—	℃
T_R	Resolution (RMS)	0 ~ 50 ℃	—	0.05	—	℃

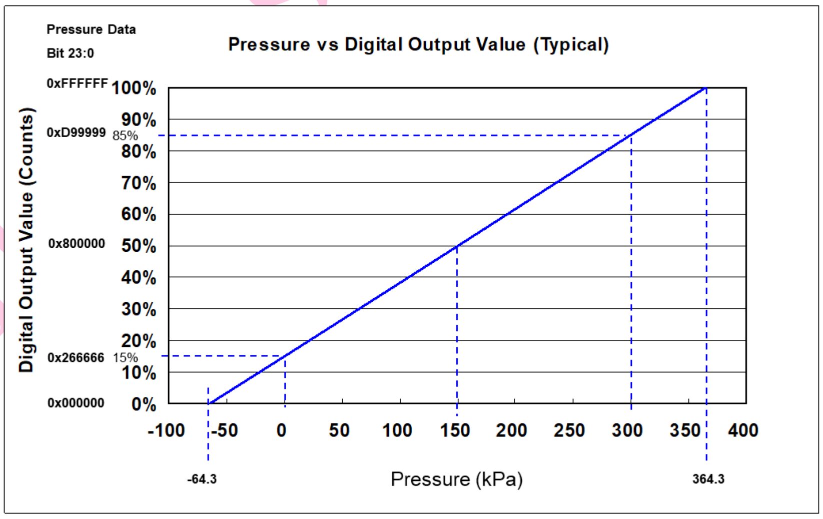
Temperature output calculation

Temperature output value can be calculated as below:

$$\text{Tout (℃)} = \frac{[\text{ Temperature Data Byte (Bit 23:0) }]\ \text{x150}}{\text{0xFFFFFF}_{HEX}} - 40$$

Pressure versus digital out value

The relationship between digital output value and pressure is given as show below:



$$\text{Pout (kPa)} = \frac{(\text{ ADC Value}_{\text{(Bit 23:0)}} - \text{0x266666}_{HEX})\ *\ (\text{0x012C}_{HEX})}{(\text{0xb33333}_{HEX})}$$

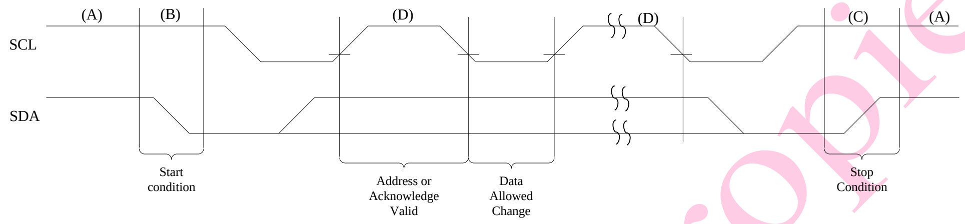
I^{2}C Operation

US6330 supports a bi-direction two wire bus and data transmission protocol to output data. A processor sends data onto the bus is defined as transmitter, US6330 receives data is defined receiver. The bus must be controlled by a master processor which generates the serial clock (SCL), controls the bus access, and generates the START and STOP conditions, while the US6330 works as slave.

The following bus protocol has been defined:

- Data transfer may be initiated only when the bus is not busy.
- During data transfer, the data line must remain stable whenever the clock is HIGH level. Changes in the data line while the clock line is HIGH will be interpreted as a START or STOP condition.

Following bus conditions has been defined



- Bus not busy as condition(A)
  Both data and clock lines remain HIGH.
- Start data transfer as condition(B)
  A HIGH to LOW transition of the SDA line while the clock (SCL) is HIGH determines s START condition. Reading data must be began by START condition.
- Stop data transfer as condition(C)
  A LOW to HIGH transition of the SDA line while the clock (SCL) is HIGH determines a STOP condition. All operation must be ended by a STOP condition.
- data valid as condition(D)
  After a START condition, the data line is stable for the duration of the HIGH period of the clock signal. The data on the line must be changed during the LOW period of the clock signal. There is one clock pulse per bit of data. The number of valid data bytes transferred between the START and STOP conditions.
- Acknowledge signal
  Each US6330 receiving, when addressed, is obliged to generate an acknowledge after the reception of each byte. The processor must generate an extra clock pulse which is associated with this acknowledge bit. The US6330 has to pull down the SDA line during the acknowledge clock pulse. The way is the SDA line is stable LOW during the HIGH period of acknowledge related clock pulse. A processor must signal an end of data to the slave by not generating an acknowledge bit on the last byte.

- US6330 control address

The seven bit is as slave address after START condition. The US6330 slave address is 1001100B (7 bits). The eighth bit of control address is read or written bit that processor wants. The processor read data sequence refers as below :

Start	Slave Address							R/W	SAK
	1	0	0	1	1	0	0	0/1	

I²C Command Format:

Writing one Byte to slave

Master	**S**	**SAD+W <1001100 0>**		**Command**		**P**
Slave (US6330)			**SAK**		**SAK**	

Notation:

*	**S**	**Start**
*	**P**	**Stop**
*	**SAD+W**	**Slave Address (1001 100) + Write bit (0)**
*	**SAD+R**	**Slave Address (1001 100) + Read bit (1)**
*	**A**	**Master acknowledge:** The microcontroller should respond a low signal to the US6330.
*	**~A**	**Master non-acknowledge:** The microcontroller should respond a high signal to the US6330.
*	**SAK**	**Slave acknowledge:** The US6330 should respond a low signal to the microcontroller.

I²C reading format for pressure data:

Example: Full measurement command (0xAA, Force Mode)

After written a command (0xAA), the master will start to read pressure value through I²C interface. Reading format as below:

Example 1: Read Pressure Data Only.

Write 0xAA Command (Force Mode)							Conversion Time Delay	Read Pressure Data											
Master	S	SAD+W <1001100 0>		Command <10101010>		P	Delay >10ms	S	SAD+R <1001100 1>		Status <Bit 7:0>	A	Pressure Data < Bit 23:16 >	A	Pressure Data < Bit 15:8 >	A	Pressure Data < Bit 7:0 >	~A	P
Slave (US6330)			SAK		SAK					SAK									

Example 2: Read Pressure and Temperature Data.

Write 0xAA Command (Force Mode)							Conversion Time Delay	Read Pressure and Temperature Data																	
Master	S	SAD+W <1001100 0>		Command <10101010>		P	Delat > 10ms	S	SAD+R <1001100 1>		Status <Bit 7:0>	A	Pressure Data < Bit 23:16 >	A	Pressure Data < Bit 15:8 >	A	Pressure Data < Bit 7:0 >	A	Temp. Data < Bit 23:16 >	A	Temp. Data < Bit 15:8 >	A	Temp. Data < Bit 7:0 >	~A	P
Slave (US6330)			SAK		SAK					SAK															

I²C Command

The command of pressure measurement as shown below:

Command	Description
0xAA$_{Hex}$ (Force Mode)	(1) When the US6330 is written a 0xAA$_{Hex}$, it will turn into force mode and start measurement pressure. (2) After pressure measurement is done, the US6330 will turn into sleep mode automatically.

NOTE:

(1) Command Delay Time:

Before next command (0xAA) write to US6330, the US6330 must has a delay time is more than 5 ms for ADC conversion time, as show below:

Command Delay Time Diagram

Command_1 (0xAA)	ADC Conversion Time Delay 5ms	Read Status and Pressure Data	Command_2 (0xAA)	ADC Conversion Time Delay 5ms	Read Status and Pressure Data	Command_3 (0xAA)	………	Command_N (0xAA)

Status Register

STATUS			
Bit	**Description**	Attr	Default
7	Reserved	R	0
6	Power supply for ADC reference voltage: 1: Power on. 0: Power off.	R	0
5	Busy: 1: Measurement is active. 0: Sleep mode (Default after POR) * This bit is reset to zero automatically after pressure sensor measurement done in force mode.	R	0
4	Reserved	R	0
3	Reserved	R	0
2	Reserved	R	0
1	Reserved	R	0
0	Reserved	R	0

Application Circuit

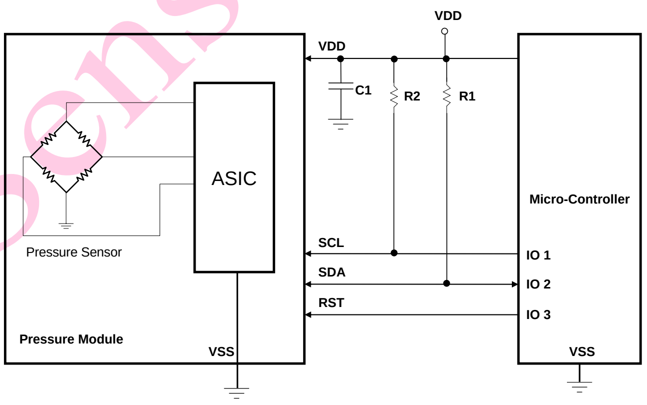


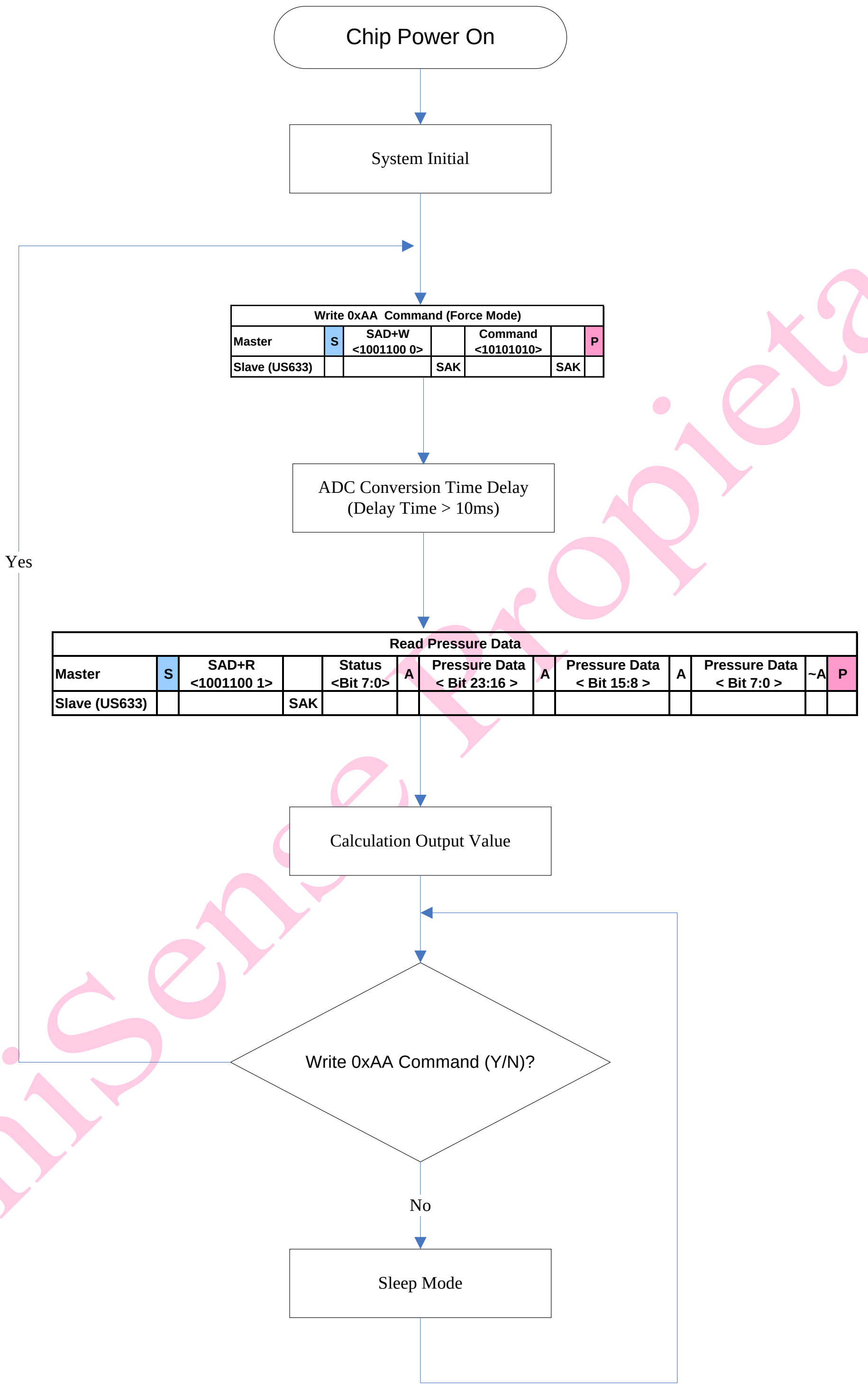
Fig. 1: I^{2}C Application Circuit.

Note:

(1) R1, R2: Pull-Up Resister. (4.7kΩ ~ 10kΩ, If needed.)

(2) C1: 0.1uF.

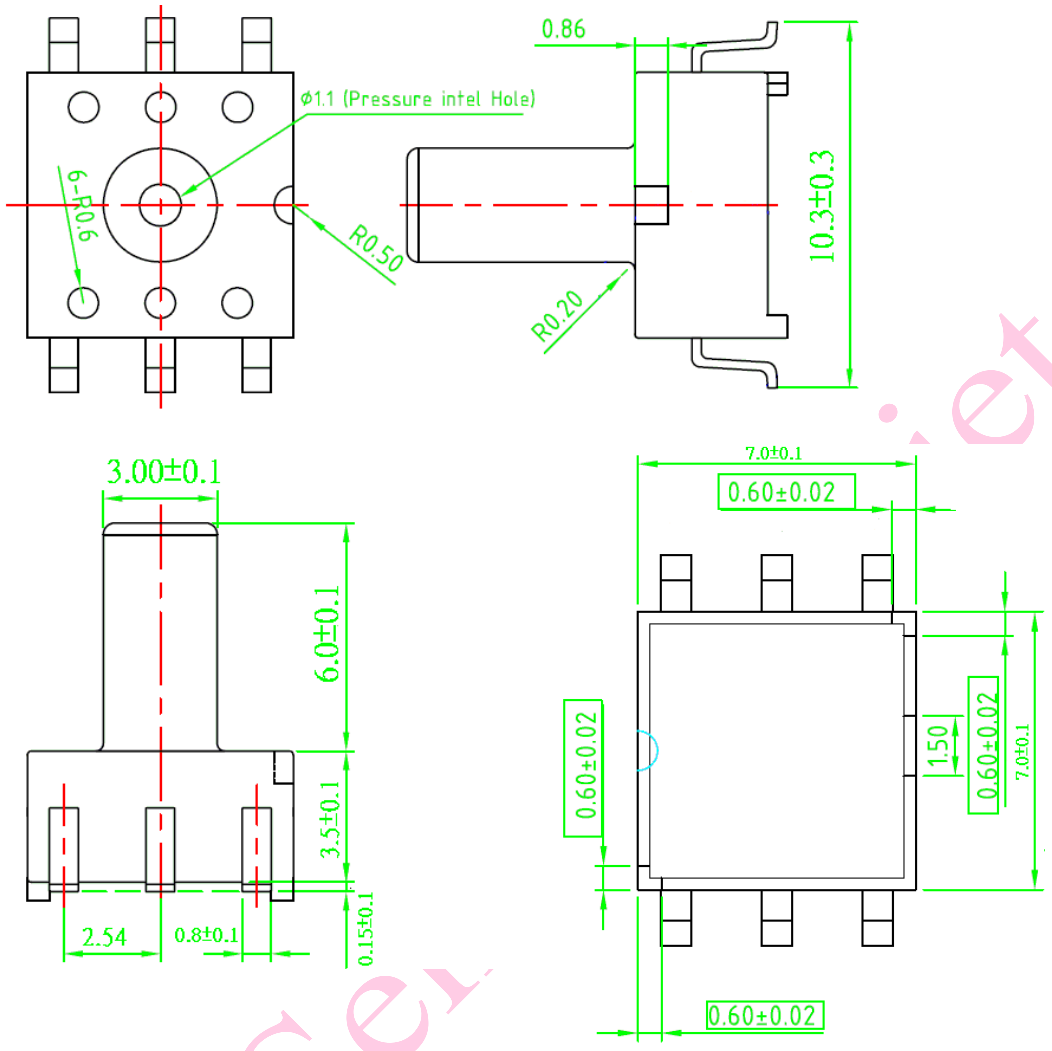
Pressure Measurement Operating Flow



Write 0xAA Command (Force Mode)						
Master	S	SAD+W <1001100 0>		Command <10101010>		P
Slave (US633)			SAK		SAK	

Read Pressure Data												
Master	S	SAD+R <1001100 1>		Status <Bit 7:0>	A	Pressure Data < Bit 23:16 >	A	Pressure Data < Bit 15:8 >	A	Pressure Data < Bit 7:0 >	~A	P
Slave (US633)			SAK									

Package Information



NOTES:
1.CONTROLLING DIMENSIONS:MM
2.MATERIAL:
a.LFAD FRAME MATERLAL: C194
b.INJECTION MOLDING MATERIAL :PPA505(BLACK)
3.DIMENSION D AND E1 DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS. MOLD FLASH OR PROTRUSIONS SHALL NOT EXCEED 0.010"[0.25mm]
4.DIMENSION "b" DOES NOT INCLUDE DAMBAR PROTRUSION.ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.003"[0.08mm]TOTAL IN EXCESS OF THE "b" DIMENSION AT MAXIMUM MATERIAL CONDITION.DAMBAR CANNOT BE LOCATED ON THE LOWER RADIUS OR THE FOOT. MINIMUM SPACE BETWEEN PTOTRUSION AND AN ADJACENT LEAD TO MB 0.0028"[0.07mm]
5.TOLERANCE: ±0.010"[0.25mm] UNLESS OTHERWISE SPECIFIED
6.OTHERWISE DIMENSION FOLLOW ACCEPTABLE SPEC.

I^{2}C Example Code

//* MAIN PRORGAM *//

```
 void main()
{
SYSTEM_INITIAL();                                   // I/O port and memory initial
ms_DELAY(5);                                         // Delay 5ms
while (1)                                           //   Read pressure loop in force mode
    {
        CMD_WRITE(0x98,0x0aa);                      // Force Mode & Full Measurement.
        m s_DELAY(10);                              //   Delay 10ms for data update time delay
        IIC_read _pressure(0x99);                 // Read Pressure data
    }
    }

//******************************Sub-Program *****************************//
void CMD_WRITE(uint16_t sub_address,uint16_t wr_data)
{
 //======= Slave Address + Bit 0 (write=0) ======================
      start();
              if((sub_address >>7) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
              if((sub_address >>6) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
              if((sub_address >>5) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
              if((sub_address >>4) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
              if((sub_address >>3) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
              if((sub_address >>2) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
              if((sub_address >>1) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
              if((sub_address >>0) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
              SACK();
          //=======write data=======================================
              if((wr_data >>7) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
              if((wr_data >>6) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
              if((wr_data >>5) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
              if((wr_data >>4) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
              if((wr_data >>3) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
              if((wr_data >>2) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
              if((wr_data >>1) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
              if((wr_data >>0) & 0x01) {SDA_DOUTSET;}
              else {SDA_DOUTCLR;} ; clock();
             SACK();
              stop();
              }
//============================================================//
```

```
void IIC_read _pressure (uint16_t    read_address)
 {
            status_reg=0; counts1=0;
            start();    // start condition
   //======= Slave Address + Bit 0(Read=1) ================================
             if((read_address >>7) & 0x01) {SDA_DOUTSET;}
            else {SDA_DOUTCLR;} ; clock();
            if((read_address >>6) & 0x01) {SDA_DOUTSET;}
            else {SDA_DOUTCLR;} ; clock();
            if((read_address >>5) & 0x01) {SDA_DOUTSET;}
            else {SDA_DOUTCLR;} ; clock();
            if((read_address >>4) & 0x01) {SDA_DOUTSET;}
            else {SDA_DOUTCLR;} ; clock();
            if((read_address >>3) & 0x01) {SDA_DOUTSET;}
            else {SDA_DOUTCLR;} ; clock();
            if((read_address >>2) & 0x01) {SDA_DOUTSET;}
            else {SDA_DOUTCLR;} ; clock();
            if((read_address >>1) & 0x01) {SDA_DOUTSET;}
            else {SDA_DOUTCLR;} ; clock();
            if((read_address >>0) & 0x01) {SDA_DOUTSET;}
            else {SDA_DOUTCLR;} ; clock();
               SACK();    // master ack low
              SDA_IN ;    // set sda input
 //=======Read status =================================================
        if (SDA==1 )   status_reg+=128;          clock();
        if(SDA==1 )   status_reg+=64;            clock();
        if (SDA==1 ) status_ reg +=32;           clock();
        if (SDA==1 )   status_ reg +=16;           clock();
        if (SDA==1 )   status_reg+=8;             clock();
        if (SDA==1 )   status_reg+=4;             clock();
        if (SDA==1 )   status_reg+=2;             clock();
        if (SDA==1 )   status_reg+=1;             clock();
             MACK();
```

```
      //=======Read pressure Data Bit 23:16 =============================================
        if (SDA==1 )   counts1+=0x800000;          clock();
        if(SDA==1 )   counts1+=0x400000;           clock();
        if (SDA==1 ))   counts1+=0x200000;          clock();
        if(SDA==1 )   counts1+=0x100000;           clock();
        if (SDA==1 )   counts1+=0x080000;            clock();
        if (SDA==1 )   counts1+=0x040000;            clock();
        if (SDA==1 )   counts1+=0x020000;            clock();
        if (SDA==1 )   counts1+=0x010000;            clock();
     //======= Read pressure Data Bit 15:8===================
      MACK();     // master ack low
        if(SDA==1 )   counts1+=0x8000;          clock();
        if (SDA==1 )   counts1+=0x4000;           clock();
        if (SDA==1 )   counts1+=0x2000;           clock();
        if (SDA==1 )   counts1+=0x1000;           clock();
        if (SDA==1 )   counts1+=0x0800;            clock();
        if (SDA==1 )   counts1+=0x0400;            clock();
        if (SDA==1 )   counts1+=0x0200;            clock();
        if (SDA==1 )   counts1+=0x0100;            clock();
         MACK();   // master ack low
//======= Read pressure Data Bit 7:0===================
        if (SDA==1 )     counts1+=0x80;          clock();
        if (SDA==1 )     counts1+=0x40;           clock();
        if (SDA==1 )     counts1+=0x20;           clock();
        if (SDA==1 )     counts1+=0x10;           clock();
        if (SDA==1 )     counts1+=0x08;            clock();
        if (SDA==1 )     counts1+=0x04;            clock();
        if (SDA==1 )     counts1+=0x02;            clock();
        if (SDA==1 ))     counts1+=0x01;            clock();
        NACK();                  // master ack High
         stop();
```

```
//===============Calculation Pressure output value ==================//
        Offset =0x266666;        // offset value
        negative_flag= 0;       // clear negative_flag
        if (counts1>= Offset)
                {
             // ====== Calculation pressure value    =====//
                    counts1= (counts1- Offset)* (0x12C)/(0xb33333); }
                else
                {
                            counts1= (Offset -counts1)*(0x12C) /(0xb33333);    // Calculation negative pressure value
                            negative_flag= 1;                                   // Negative pressure Flag is set.
                }
               Display_Pressure(counts1, negative_flag);        // display pressure value
      }
```

```
void start()     // start condition //
{
SDA_OUT;                  // SDA set to Output mode.
SDA_DOUTSET;              // SDA Output high.
SCL_DOUTSET;          // SCL Output high.
NOP_DELAY(30);
SDA_DOUTCLR;          // SDA output low.
NOP_DELAY(30);
SCL_DOUTCLR;       // SCL output low.
NOP_DELAY(30);
}

void stop()
{
SDA_OUT;              // SDA set to Output mode.
SDA_DOUTCLR;
SCL_DOUTCLR;
NOP_DELAY(30);
SCL_DOUTSET;
NOP_DELAY(30);
SDA_DOUTSET;
NOP_DELAY(30);
}

void clock()
{
  SCL_DOUTSET;
  NOP_DELAY(2);
  NOP_DELAY(2);
  SCL_DOUTCLR;
  NOP_DELAY(1);
  NOP_DELAY(1);
}
```

```
void MACK()
{
 SDA_OUT ;              // SDA set to output mode
 SDA_DOUTCLR;
   NOP_DELAY(30);
 clock();
   NOP_DELAY(30);
 SDA_IN ;   // set sda input
   NOP_DELAY(30);
}

void NACK()
{
 SDA_OUT ;            // SDA set to output mode
 SDA_DOUTSET;      // SDA output high
 NOP_DELAY(30);
clock();
   NOP_DELAY(30);
}

void SACK()
{
 SDA_IN ;     // SDA set to input mode
 NOP_DELAY(30);
 SCL_DOUTSET;
 NOP_DELAY(30);
 ack_error_flag=SDA_N;     // read ACK Signal
   clock();
 NOP_DELAY(30);
 SDA_OUT;
 NOP_DELAY(30);
   }
//======================End Of I2C Example Code ===============================///
```