Key Features

- Piezo-resistive silicon micro-machined sensor
- Gauge type pressure sensor
- I^2C / SPI Interface
- Pressure range: -15 to +150 psi
- Pressure Sensitivity: 0.0002 psi/LSB
- 24 Bit Σ-Δ ADC
- Temperature Compensation: 0 ~ 50 ^oC
- Operating voltage 3.0V
- Operating mode current: ~0.6mA (typical)
- Sleep Mode current: 20nA (typical)
- SOP6 package
- RoHS compliant and Halogen-free

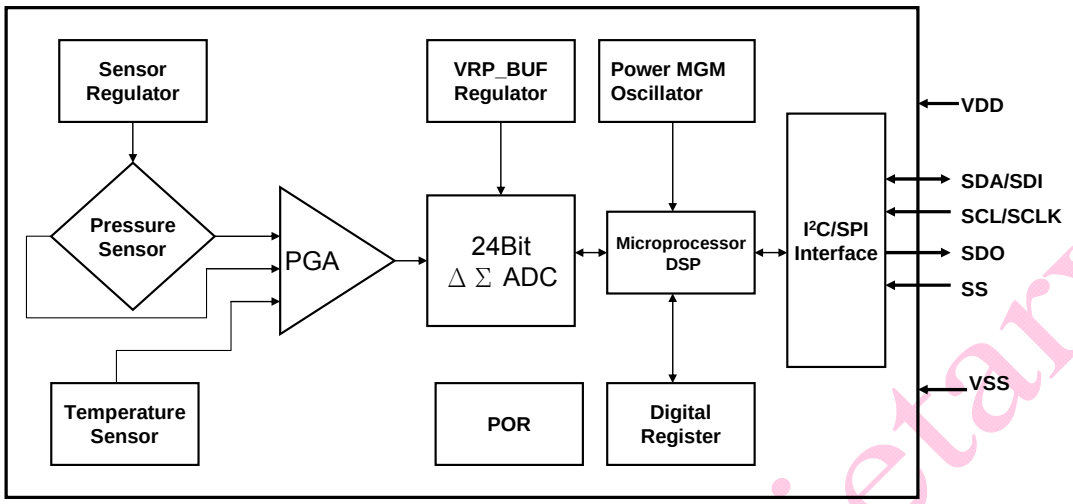
Applications

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

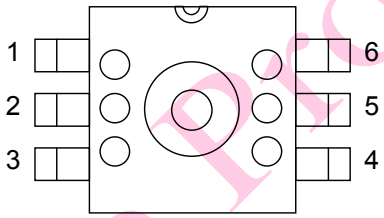
Description

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	P	Connected to GND
2	VDD	P	Positive supply voltage
3	SS	I	For I²C/SPI mode selection (SS: internal is pull-up) : (1) If the SS pin is high level or floating status after power on, and first command is I²C format, then the communication interface will be operated in I²C mode. (2) If the SS pin is low level after power on, the communication interface will be SPI mode.
4	SDA/SDI	I/O	(1) Data in/out for I²C. (2) Data input for SPI.
5	SCL/SCLK	I	Clock input for I²C/SPI
6	SDO	O	Data output for SPI.

Maximum Ratings

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.

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	—	—	—	+450	psi
P_{OP}	Operating pressure	—	-15	—	+150	psi
P_{RACC}	Relative Accuracy	P: -15 ~ +150 psi , T: 25℃	—	±0.3	—	psi
P_{AACC1}	Absolute Accuracy1	P: -15 ~ +100 psi, T: 0 ~ +50 ℃	-1.0	—	+1.0	psi
P_{AACC2}	Absolute Accuracy2	P: -15 ~ +150 psi, T: 0 ~ +50 ℃	-1.5	—	+1.5	psi
P_{RES}	ADC Resolution	By internal option	—	20	—	Bit
P_{SEN}	Pressure Sensitivity	—	—	0.0002	—	psi/LSB
P_{NS}	Pressure Resolution (RMS)	—	—	0.002	—	psi

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	—	7	—	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}	I²C Clock Frequency	—	—	—	3.4	MHz
F_{SPI}	SPI Clock Frequency	—	—	1	20	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:

$$\text{Tout}(℃) = \frac{[\text{ Temperature Data Byte (Bit 23:0) }] \times 150}{0\text{xFFFFFF}_{HEX}} - 40$$

I²C Operation

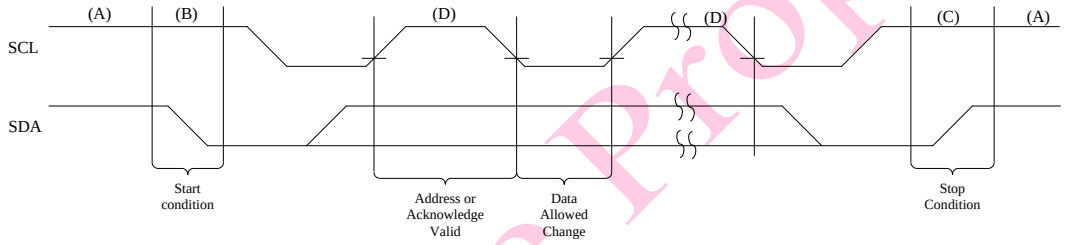
The SS must be high after power on, and the first command must be I2C command, the I²C mode will be selected.

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 output 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									

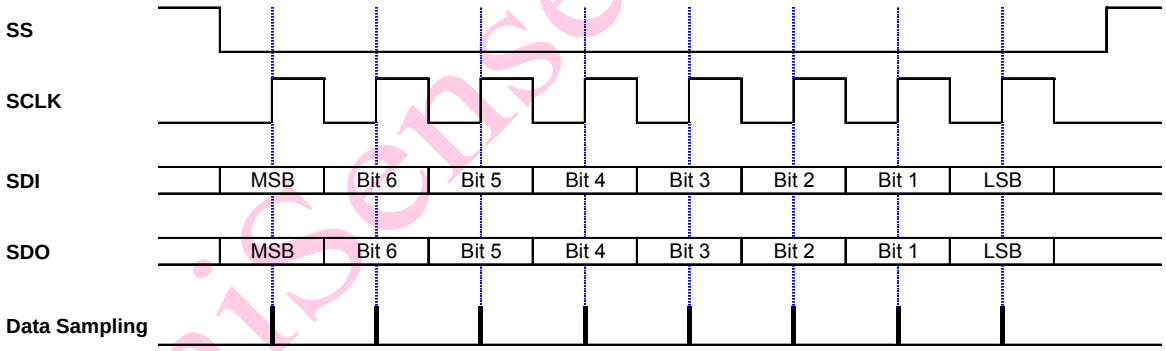
Example 2: Read Pressure and Temperature Data.

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															

SPI Operation

When the SS pin is falling to low, the SPI mode will be selected. The processor can read digital data through SPI communication interface.

In clock-edge, if SCLK is idle state that will output low. And data latch with rising-edge, data output with falling-edge. The SPI sequence as below:



SPI Command Request

The command request must be 3 bytes, command format as below:

SDI	Command	$0x00_{Hex}$	$0x00_{Hex}$
SDO	Status < Bit 7:0 >	Data < Bit 15:8 >	Data < Bit 7:0 >

SPI Read Request

SDI	Command (NOP) <0x00>	Command Data < Bit 15:8 >	Command Data < Bit 7:0 >
SDO	Status < Bit 7:0 >	Data < Bit 15:8 >	Data < Bit 7:0 >

SPI Read Example:

Example1: Reading pressure data with measurement command ($0xAA_{Hex}$) as below SPI format:

Pin Name	Force Mode Command Format			Conversion Time Delay	Read Pressureand and Temperature Data			
SDI	Command <0xAA>	$0x00_{Hex}$	$0x00_{Hex}$	Delay Time>10ms	Command (NOP) <0x00>	$0x00_{Hex}$	$0x00_{Hex}$	$0x00_{Hex}$
SDO	Status < Bit 7:0 >	Data < Bit 15:8 >	Data < Bit 7:0 >		Status < Bit 7:0 >	Pressure Data < Bit 23:16 >	Pressure Data < Bit 15:8 >	Pressure Data < Bit 7:0 >

SS	SS=Low (SPI Enable)	SS=High (SPI Disable)	SS=Low (SPI Enable)	SS=High (SPI Disable)

Example2: Reading pressure and temperature data with measurement command ($0xAA_{Hex}$) as below SPI format:

Pin Name	Force Mode Command Format			Conversion Time Delay	Read Pressureand and Temperature Data						
SDI	Command <0xAA>	$0x00_{Hex}$	$0x00_{Hex}$	Delay Time>10ms	Command (NOP) <0x00>	$0x00_{Hex}$	$0x00_{Hex}$	$0x00_{Hex}$	$0x00_{Hex}$	$0x00_{Hex}$	$0x00_{Hex}$
SDO	Status < Bit 7:0 >	Data < Bit 15:8 >	Data < Bit 7:0 >		Status < Bit 7:0 >	Pressure Data < Bit 23:16 >	Pressure Data < Bit 15:8 >	Pressure Data < Bit 7:0 >	Temp. Data < Bit 23:16 >	Temp. Data < Bit 15:8 >	Temp. Data < Bit 7:0 >

SS	SS=Low (SPI Enable)	SS=High (SPI Disable)	SS=Low (SPI Enable)	SS=High (SPI Disable)

I²C/SPI 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 10 ms for ADC conversion time, as show below:

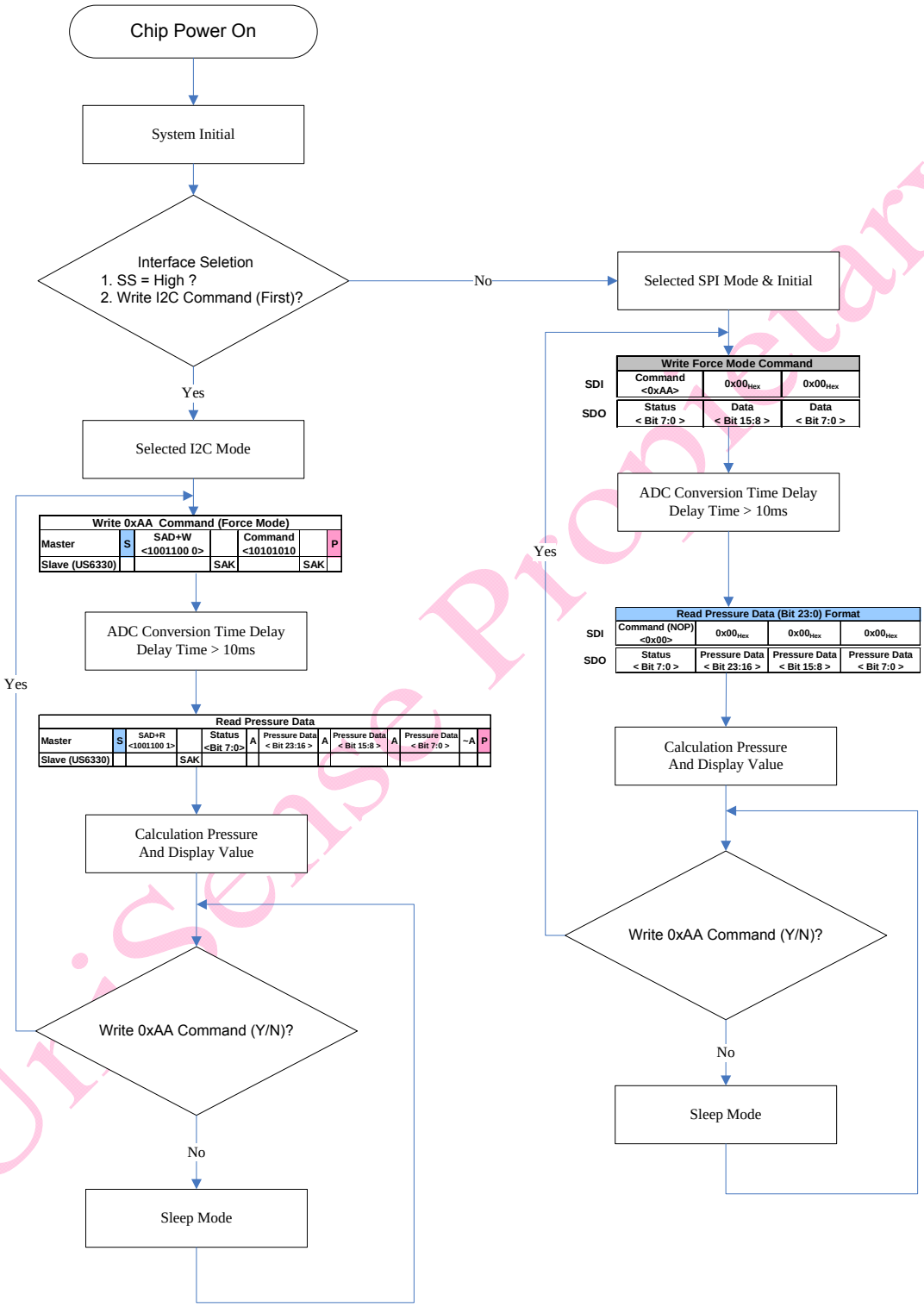
Command Delay Time Diagram

Command_1 (0xAA)	ADC Conversion Time Delay 10ms	Read Status and Pressure Data	Command_2 (0xAA)	ADC Conversion Time Delay 10ms	Read Status and Pressure Data	Command_3 (0xAA)	.....	Command_N (0xAA)

Status Register

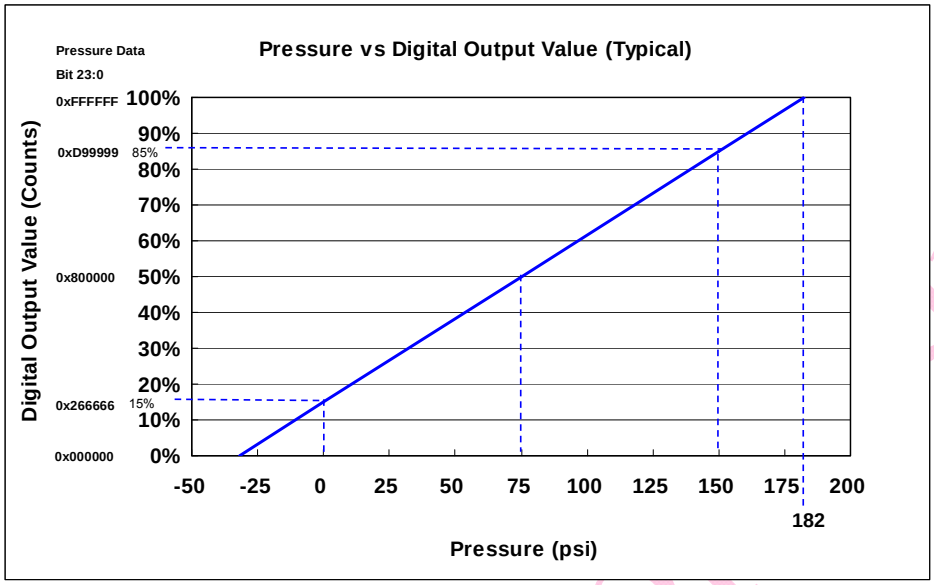
STATUS			
Bit	**Description**	Attr	Default
7	Reserved	R	0
6	Power supply for ADC reference voltage: 1: Power on. 0: Power off.	R	0
5	Busy: 1: Measurement is active. 0: Sleep mode (Default after POR) * This bit is reset to zero automatically after pressure sensor measurement done in force mode.	R	0
4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

Pressure Measurement Operating Flow



Pressure versus digital out value

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



$$\text{Pout (psi)} = \frac{(\text{ADC Value}_{(\text{Bit 23:0})} - 0\text{x266666}_{\text{HEX}}) * (0\text{x96}_{\text{HEX}})}{(0\text{xb33333}_{\text{HEX}})}$$

Application Circuit (For I²C/SPI Interface)

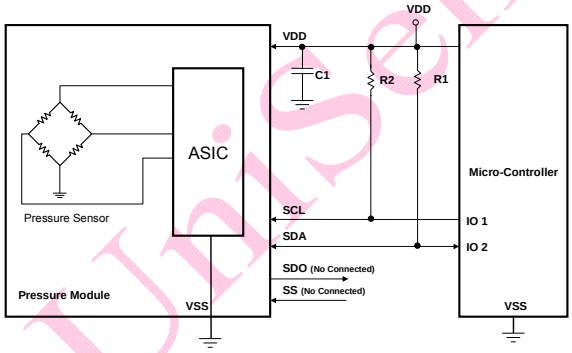


Fig. 1: I²C Application Circuit.

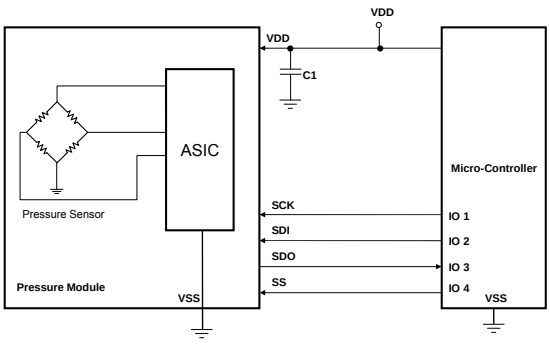


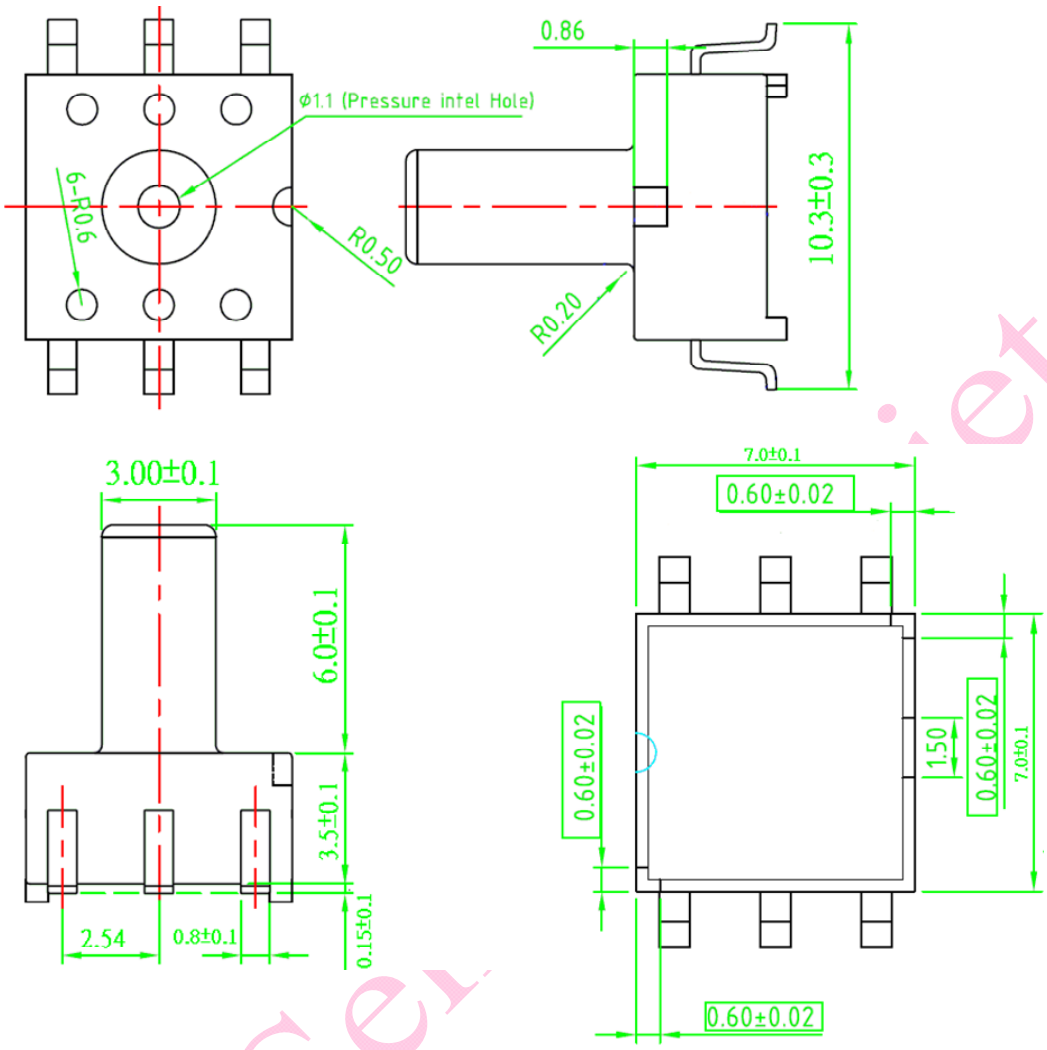
Fig. 2: SPI Application Circuit.

Note:

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

(2) C1: 0.1uF.

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.

I²C 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)* (0x96)/(0xb33333); }
            else
            {
                    counts1= (Offset -counts1)*(0x96) /(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 ================================///
```

SPI Example Code

//* MAIN PRORGAM *//

```
void main()
{
SYSTEM_INITIAL();                                          // I/O port and memory initial
ms_DELAY(5);                                               // Delay 5ms
while (1)
{
//******************************** Main Loop for SPI Read Pressure ********************************************//
        spi_command(0xaa,0x00 );                  // Write a force mode Command (0xAA)
       ms_DELAY(10);                               // Delay 10ms for data update time delay
        spi_read(0x00,0x00 );                                  // Read Pressure
 //************************************* SPI Read end ***************************************************************//
}
}



//*******************************SPI Sub-Program ******************************//
void clock()
{
 SCL_DOUTSET;              // SCLK Output High
 NOP_DELAY(2);
 SCL_DOUTCLR;              // SCLK Output Low
 NOP_DELAY(2);
}
```

```
void spi_command(uint16_t command, uint16_t command_data )
 {
    //=========== IO & Register initail ==================================
                SS_DOUTSET;                  // SS Output High.
                SDI_OUT;                    // SDI set to Output Mode
                SDI_DOUTCLR;                 // SDI Output Low
                SCLK_DOUTCLR;                // SCLK Output Low
                SS_DOUTCLR;                  // SS Output Low
                status_reg=0;                // Status Clear
 //=======Write command & Read Status Register=================================
                if((command>>7) & 0x01) {SDI_DOUTSET;}
                else
                { SDI_DOUTCLR;} SCLK_DOUTSET;
                if (SDO) status_reg+=0x0080;
                clock();

                if((command>>6) & 0x01) {SDI_DOUTSET;}
                else {SDI_DOUTCLR;}     SCLK_DOUTSET;
                  if (SDO) status_reg+=0x0040;
               clock();

                if((command>>5) & 0x01) {SDI_DOUTSET;}
                else {SDI_DOUTCLR;}      SCLK_DOUTSET;
                  if (SDO) status_reg+=0x0020;
                clock();


                if((command>>4) & 0x01) {SDI_DOUTSET;}
                else {SDI_DOUTCLR;}     SCLK_DOUTSET;
                  if (SDO) status_reg+=0x0010;
                clock();

                if((command>>3) & 0x01) {SDI_DOUTSET;}
                else {SDI_DOUTCLR;}      SCLK_DOUTSET;
                  if (SDO) status_reg+=0x0008;
                clock();
```

```
        if((command>>2) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;}     SCLK_DOUTSET;
        if (SDO) status_reg+=0x0004;
        clock();

        if((command>>1) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;}       SCLK_DOUTSET;
        if (SDO) status_reg+=0x0002;
        clock();

        if((command>>0) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;}     SCLK_DOUTSET;
        if (SDO) status_reg+=0x0001;
        clock();

//================= Write command DATA    High Byte ================================
        if((command_data >>15) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;} ; clock();
        if((command_data >>14) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;} ; clock();
        if((command_data >>13) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;} ; clock();
        if((command_data >>12) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;} ; clock();



        if((command_data >>11) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;} ; clock();
        if((command_data >>10) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;} ; clock();
        if((command_data >>9) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;} ; clock();
        if((command_data >>8) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;} ; clock();
```

```
//=========== Write command DATA   Low Byte ==========================//
                if((command_data >>7) & 0x01) {SDI_DOUTSET;}
               else {SDI_DOUTCLR;} ; clock();
               if((command_data >>6) & 0x01) {SDI_DOUTSET;}
               else {SDI_DOUTCLR;} ; clock();
               if((command_data >>5) & 0x01) {SDI_DOUTSET;}
               else {SDI_DOUTCLR;} ; clock();
               if((command_data >>4) & 0x01) {SDI_DOUTSET;}
               else {SDI_DOUTCLR;} ; clock();

               if((command_data >>3) & 0x01) {SDI_DOUTSET;}
               else {SDI_DOUTCLR;} ; clock();
               if((command_data >>2) & 0x01) {SDI_DOUTSET;}
               else {SDI_DOUTCLR;} ; clock();
               if((command_data >>1) & 0x01) {SDI_DOUTSET;}
               else {SDI_DOUTCLR;} ; clock();
               if((command_data >>0) & 0x01) {SDI_DOUTSET;}
               else {SDI_DOUTCLR;} ; clock();
//================= When Write command is done, then the SS Output High ====================
               SS_DOUTSET;                  // SS output High
               }


void spi_read(uint16_t command, uint16_t command_data )
       {
               status_reg=0;
               pressure_data=0;
       //=========== IO & Register initail ===================================
               SS_DOUTSET;                  // SS Output High.
               SDI_OUT;                    // SDI set to Output Mode
               SDI_DOUTCLR;                 // SDI Output Low
               SCLK_DOUTCLR;                // SCLK Output Low
               SS_DOUTCLR;                  // SS Output Low
               status_reg=0;                // Status Clear
```

```
//======= Read Pressure Command (NOP) ================================
        if((command >>7) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;}    SCLK_DOUTSET;
        if (SDO)    status_reg+=0x0080;
        clock();

        if((command >>6) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;}    SCLK_DOUTSET;
        if (SDO)    status_reg+=0x0040;
        clock();

        if((command >>5) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;}    SCLK_DOUTSET;
        if (SDO)    status_reg+=0x0020;
        clock();

        if((command >>4) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;}     SCLK_DOUTSET;
        if (SDO)    status_reg+=0x0010;
        clock();

        if((command >>3) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;}     SCLK_DOUTSET;
        if (SDO)    status_reg+=0x008;
        clock();
        if((command >>2) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;}     SCLK_DOUTSET;
        if (SDO)    status_reg+=0x004;
        clock();

        if((command >>1) & 0x01) {SDI_DOUTSET;}
        else {SDI_DOUTCLR;}     SCLK_DOUTSET;
        if (SDO)    status_reg+=0x002;
        clock();
```

```
if((command >>0) & 0x01) {SDI_DOUTSET;}
else {SDI_DOUTCLR;}     SCLK_DOUTSET;
if (SDO)    status_reg+=0x001;
clock();

//=================== Read Pressure Data (Bit 23:16) ==========================//
if((command_data >>15) & 0x01) {SDI_DOUTSET;}
else {SDI_DOUTCLR;}        SCLK_DOUTSET;
if(SDO)    pressure_data +=0x800000;
clock();

if((command_data >>14) & 0x01) {SDI_DOUTSET;}
else {SDI_DOUTCLR;}       SCLK_DOUTSET;
if (SDO) pressure_data +=0x400000;
clock();

if((command_data >>13) & 0x01) {SDI_DOUTSET;}
else {SDI_DOUTCLR;}        SCLK_DOUTSET;
if (SDO) pressure_data +=0x200000;
clock();

if((command_data >>12) & 0x01) {SDI_DOUTSET;}
else {SDI_DOUTCLR;}     SCLK_DOUTSET;
if (SDO) pressure_data +=0x100000;
clock();

if((command_data >>11) & 0x01) {SDI_DOUTSET;}
else {SDI_DOUTCLR;}       SCLK_DOUTSET;
if (SDO) pressure_data +=0x80000;
clock();

if((command_data >>10) & 0x01) {SDI_DOUTSET;}
else {SDI_DOUTCLR;}     SCLK_DOUTSET;
if (SDO) pressure_data +=0x40000;
clock();
```

```
            if((command_data >>9) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}        SCLK_DOUTSET;
            if (SDO) pressure_data +=0x20000;
            clock();

            if((command_data >>8) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}      SCLK_DOUTSET;
            if (SDO) pressure_data +=0x10000;
            clock();

//=================== Read Pressure Data (Bit 15:8) ========================//
            if((command_data >>15) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}        SCLK_DOUTSET;
            if(SDO)    pressure_data +=0x8000;
            clock();

            if((command_data >>14) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}      SCLK_DOUTSET;
            if (SDO) pressure_data +=0x4000;
            clock();

            if((command_data >>13) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}      SCLK_DOUTSET;
            if (SDO) pressure_data +=0x2000;
            clock();

            if((command_data >>12) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}     SCLK_DOUTSET;
            if (SDO) pressure_data +=0x1000;
           clock();

            if((command_data >>11) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}      SCLK_DOUTSET;
            if (SDO) pressure_data +=0x0800;
            clock();
```

```
            if((command_data >>10) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}     SCLK_DOUTSET;
            if (SDO) pressure_data +=0x0400;
            clock();

            if((command_data >>9) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}        SCLK_DOUTSET;
            if (SDO) pressure_data +=0x0200;
            clock();

            if((command_data >>8) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}       SCLK_DOUTSET;
            if (SDO) pressure_data +=0x0100;
            clock();
//=================== Read Pressure Data (Bit 7:0) ========================//
            if((command_data >>7) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}     SCLK_DOUTSET;
            if (SDO) pressure_data +=0x0080;
            clock();

            if((command_data >>6) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}       SCLK_DOUTSET;
            if (SDO) pressure_data +=0x0040;
            clock();


            if((command_data >>5) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}     SCLK_DOUTSET;
            if (SDO) pressure_data +=0x0020;
            clock();

            if((command_data >>4) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}       SCLK_DOUTSET;
            if (SDO) pressure_data +=0x0010;
            clock();
```

```
            if((command_data >>3) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}        SCLK_DOUTSET;
            if (SDO) pressure_data +=0x0008;
            clock();
            if((command_data >>2) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}     SCLK_DOUTSET;
            if (SDO) pressure_data +=0x0004;
            clock();
            if((command_data >>1) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}       SCLK_DOUTSET;
            if (SDO) pressure_data +=0x0002;
            clock();
            if((command_data >>0) & 0x01) {SDI_DOUTSET;}
            else {SDI_DOUTCLR;}       SCLK_DOUTSET;
            if (SDO) pressure_data +=0x0001;
            clock();
      //================ Read pressure is done , then SS output High ==============================
             SS_DOUTSET;           // SS output High for end
      //===============Calculation Pressure output value ==================//
      Offset =0x266666;         // offset value
      negative_flag= 0;       // clear negative pressure flag
      if (pressure_data >= Offset)
            {
          // ====== Calculation pressure value    =====//
             pressure_data = (pressure_data - Offset)* (0x96)/(0xb33333);
            }
           else
           {
        pressure_data =(Offset - pressure_data)*( 0x96)/(0xb33333);    // Calculation negative pressure value
        negative_flag= 1;                                                    // Negative pressure flag is set
           }
          Display_Pressure(pressure_data, negative_flag);                    // display pressure value
           }
           }
//=======================End Of SPI Example Code================================///
```