PS-200D with cover

Differential Pressure Sensor With I²C Digital Output

- Piezo-resistive silicon micro-machined sensor
- Operating voltage : 3.0V
- Digital output : I²C interface
- Pressure range : -1,000 to +1,000 Pa
- Operating mode current : ~0.6mA (typical)
- 24bit ADC
- RoHS compliant and Halogen-free Package
- Approximately 200Hz data output rate
- Available package : SOP6

PS-200D with cover

TYPICAL APPLICATIONS

- ✓ Medical instrumentation
- ✓ Pneumatic control
- ✓ Air cleaner
- ✓ Filter Control

❒ Electrical Characteristics

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage	—	—	1.68	3.0	3.6	V
I_{DD}	Operating Current	3.0V	—	—	0.6	1.2	mA
I_{STB1}	Standby Current	3.0V	Temperature ≦ 85℃	—	20	250	nA
P_{PRO}	Proof pressure	—	—	—	—	+20,000	Pa
P_{OP}	Operating pressure	3.0V	—	-1,000	—	+1,000	Pa
P_{RACC}	Relative Accuracy	3.0V	P: +/- 1000 Pa, T: 0 ~ +50℃	-10	—	+10	pa
P_{AACC1}	Absolute Accuracy 1	3.0V	P: +/- 1000 Pa, T: +25 ℃	-20	—	+20	Pa
P_{AACC2}	Absolute Accuracy 2	3.0V	P: +/- 1000 Pa, T: 0 ~ +50 ℃	-30	—	+30	Pa
P_{RES}	ADC Resolution	—	By internal option	—	20	—	bit
P_{SEN}	Pressure Sensitivity	3.0V	—	—	0.0027	—	Pa/LSB
P_{NS}	Pressure Resolution (RMS)	3.0V	—	—	0.1	—	Pa
t_{ADC}	AD Conversion Time	3.0V	—	—	1.7	—	ms
t_{update}	Data Update Time	3.0V	Full measurement and tem perature compensation	—	7	—	ms
t_{ST}	Start-Up Time	3.0V	VDD ramp up to communication	—	—	1	ms
			VDD ramp up to Analog operation	—	—	2.5	ms
t_{SLP}	Wake-UP Time	3.0V	Sleep to communication	—	—	0.5	ms
			Sleep to analog operation	—	—	2	ms
Fsys	System Frequency	3.0V	—	—	4	—	MHz
F_{I2C}	I²C Clock Frequency	3.0V	—	—	—	3.4	MHz
F_{SPI}	SPI Clock Frequency	3.0V	—	—	1	20	MHz

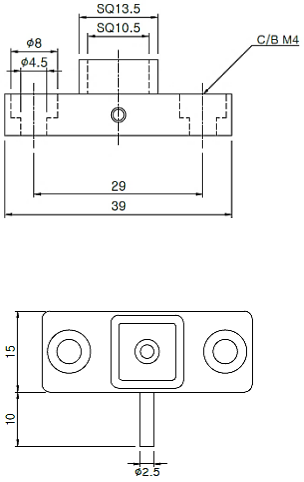
❒ Maximum Ratings

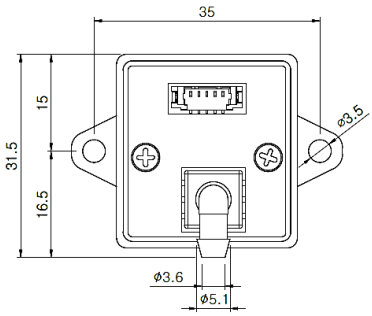
Maximum Ratings	
Voltage (with respect to GND unless otherwise noted)	
VDD	-0.4 V to +3.63 V
Voltage at Digital IO Pins	-0.5V to VDD+0.5 V
Operating Temperature Range	-40 to +85℃
Storage Temperature Range	-40 to +125℃
Electrostatic Discharge Tolerance - Human Body Model	+/- 2KV

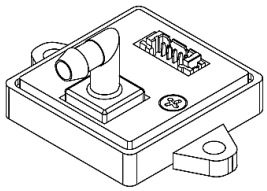
❒ Dimensions (mm)

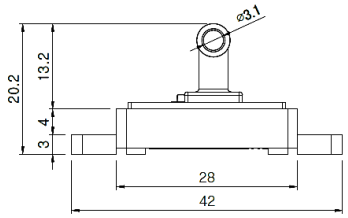
Dimensions (mm)

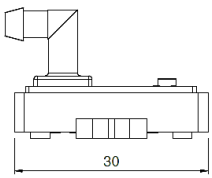
Bracket

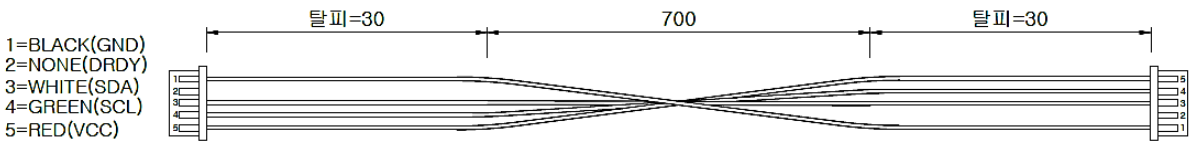








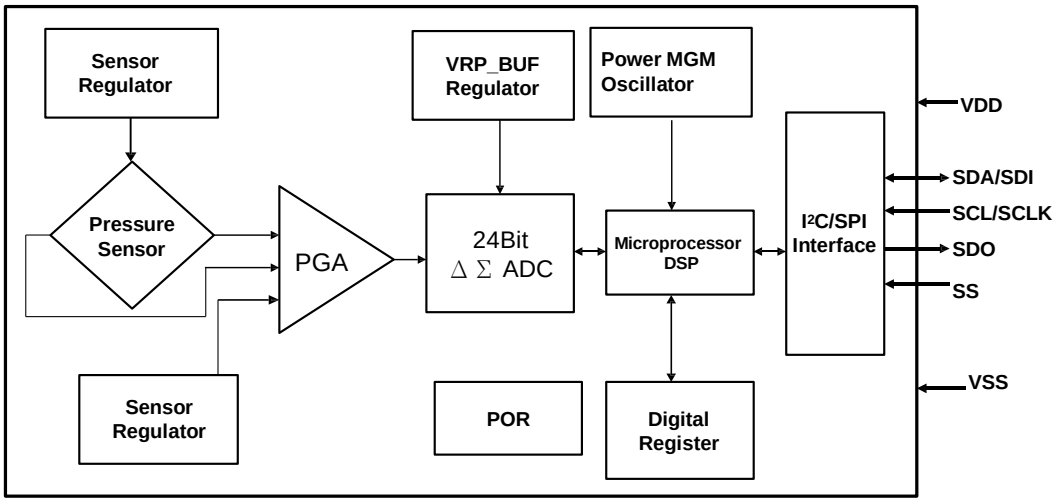




HOUSING : 12505HS-05(1.25mm pitch 5pin)
수축튜브 : 각 20mm

❒ Main ASIC Part Diagram

Diagram



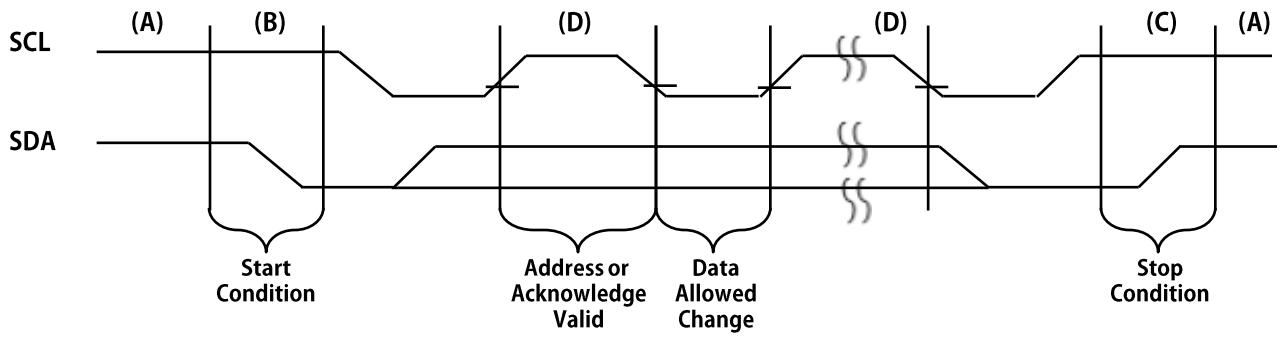
❒ System Reading Timing

Chart

The SS must be high after power on, and the first command must be I²C command, the I²C mode will be selected. PS-200D 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, PS-200D 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 PS-200D 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 keep 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.

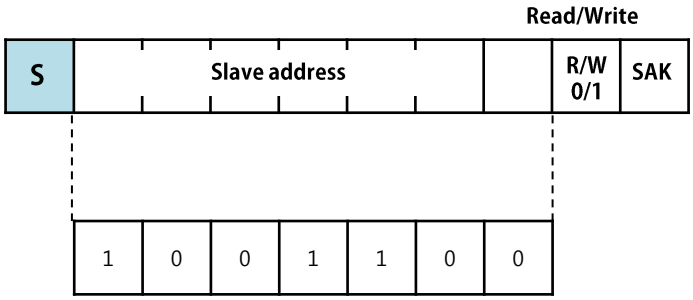
❖ Data valid as condition(D)
After a START condition, the data line is stable for the duration of the HIGH period of the clock signal. The data on the line must be changed during the LOW period of the clock signal. There is one clock pulse per bit of data. The number of valid data bytes transferred between the START and STOP conditions.

❖ Acknowledge signal Each PS-200D 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 PS-200D 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.

❒ PS-200D control address

chart

The seven bit is as slave address after START condition. The PS-200D 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 :



I²C Command Format

Writing one Byte to slave

Master	S	SAD+W <1001100 0>		Command		P
Slave (PS-200D)			SAK		SAK	

S	START	A	Master acknowledge
P	STOP	~A	Master non-acknowledge
SAD+W	Slave Address (1001 100) + Write bit (0)	SAK	Slave acknowledge
SAD+R	Slave Address (1001 100) + Read bit (1)		

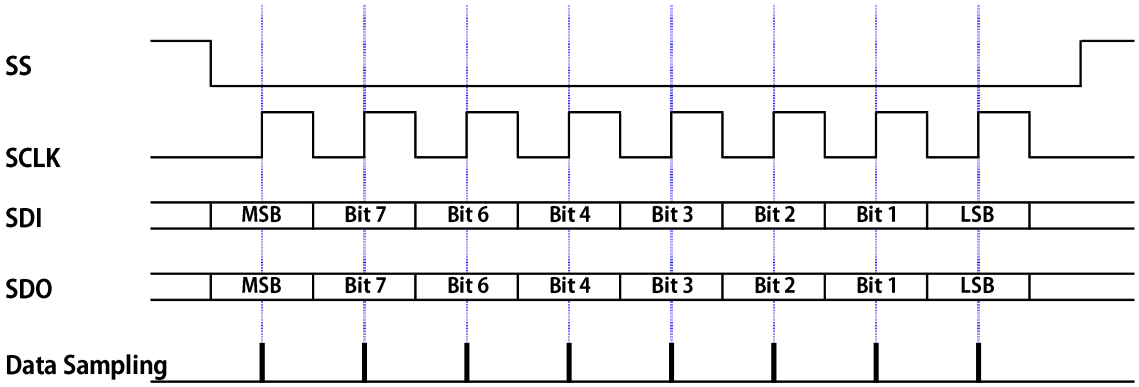
I²C reading format for pressure data:
Example: Full measurement command (0xAA, Force Mode)
After written a command (0xAA), the master will start to read pressure value through I²C interface. Reading format as below:

	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 (PS-200D)			SAK		SAK					SAK										

❒ SPI Operation

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:

Reading pressure data with measurement command ($0xAA_{Hex}$) as below I^2C format:

Pin Name	Force Mode Command Format			Conversion Time Delay	Read Pressure Data (Bit 23:0) Format			
SDI	Command <0xAA>	$0x00_{Hex}$	$0x00_{Hex}$	Delay > 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 >

I²C/SPI Command

❒ I²C/SPI Command

The command of pressure measurement as shown below:

Command	Description
$0xAA_{Hex}$ (Force Mode)	(1) When the PS-200D is written a $0xAA_{Hex}$, it will turn into force mode and start measurement pressure. (2) After pressure measurement is done, the PS-200D will turn into sleep mode automatically.

NOTE:

(1) Command Delay Time:

Before next command (0xAA) write to PS-200D, the PS-200D must has a delay time is more than 10 ms for ADC conversion time, as show below:

❒ Command Delay Time Diagram

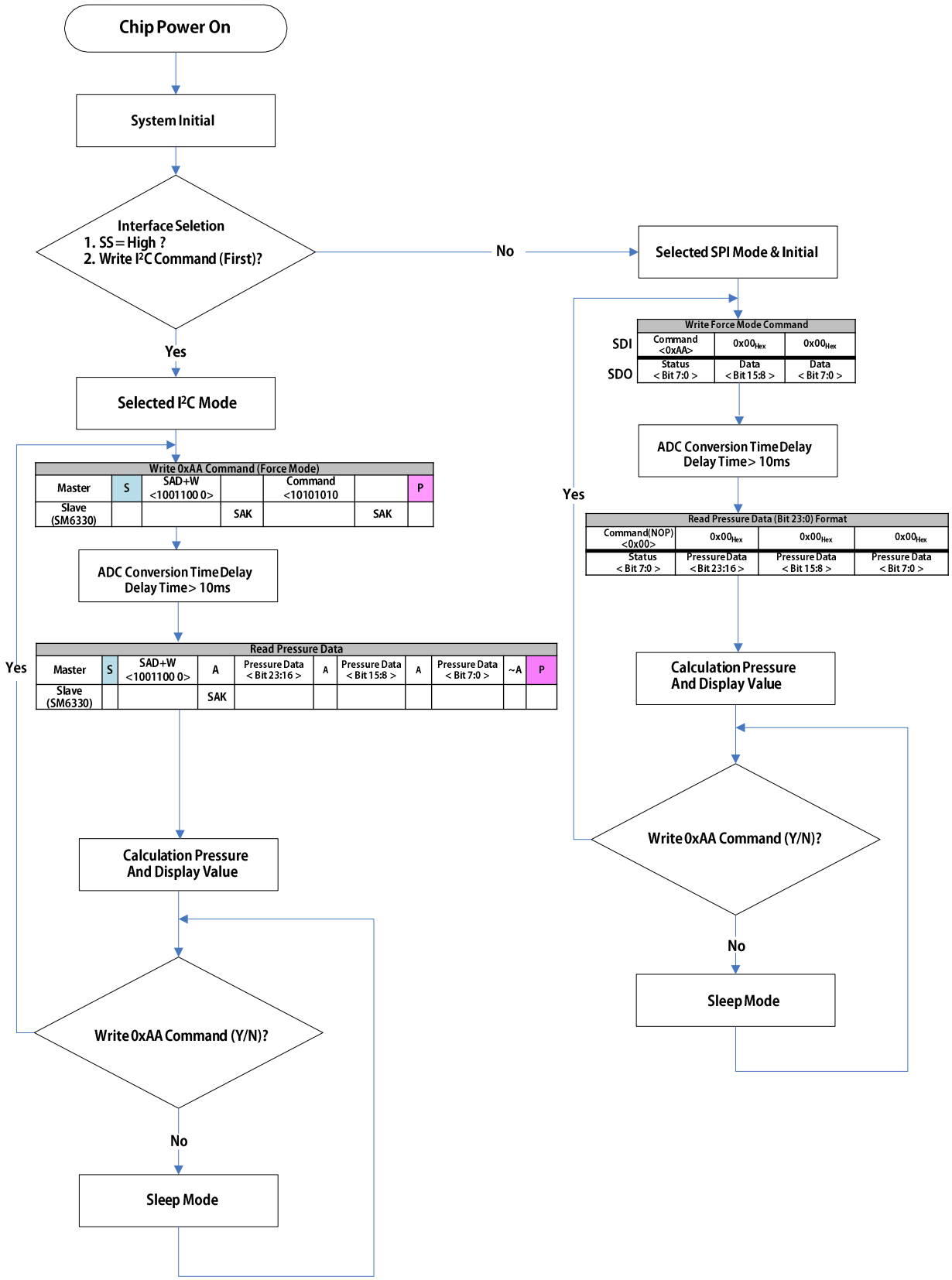
Command_1 (0xAA)	ADC Conversion Time Delay 10ms	Read Status and Pressure Data	Command_2 (0xAA)	ADC Conversion Time Delay 10ms	Read Status and Pressure Data	Command_3 (0xAA)	.....	Command_N (0xAA)

❒ 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

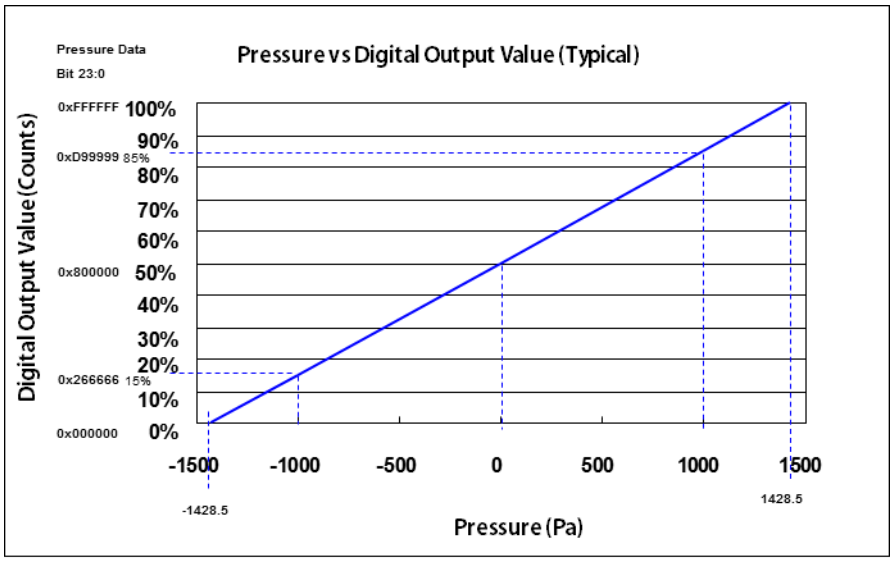
Chart



❒ Pressure versus digital out value

chart

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



$$\text{Pout (Pa)} = \frac{(\text{ADC Value}_{(\text{Bit 23:0})} - 0\text{x800000}_{\text{HEX}}) * (0\text{x07d0}_{\text{HEX}})}{(0\text{xb33333}_{\text{HEX}})}$$

❒ I²C Reading Code Example

```
//* MAIN PRORGAM *//

//* MAIN PRORGAM *//


void main()
{
SYSTEM_INITIAL(); // IO(I2C Mode: SS Pin Output High) , LCM Display, Memory Initail
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); // ADC Conversion Time Delay 10ms
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();
}
//=============================================================//
```

❒ I²C Reading Code Example

```
//* MAIN PRORGAM *//

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
```

❒ I²C Reading Code Example

```
//* MAIN PRORGAM *//

//======= 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 =0x800000; // offset value
negative_flag= 0; // clear negative_flag
if (counts1>= Offset)
{
// ====== Calculation pressure value , Display resolution : 0.1Pa =====//
counts1= (counts1- Offset)* (0x07d0)*10/(0xb33333); }
else
{
counts1= ~(( Offset -counts1)*(0x07d0)*10/(0xb33333))+1; // Calculation negative pressure value
negative_flag= 1;
}
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);
}
```

❒ I²C Reading Code Example

```
//* MAIN PRORGAM *//

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 ================================///
```