DIFFERENTIAL PRESSURE SENSOR | DIFF. Pressure Sensor [Analog Type]

Superb Sensor

APS-35V

Differential Pressure Sensor

Analog Output

- Pressure Range : 0 to 200···1,000Pa
- Piezo-resistive silicon micro-machined sensor
- Analog output : 0.5 to 2.5V or 1 to 3V
- Operating voltage : 3.0V or 5.0V
- Embedded temperature compensation
- RoHS compliant and Halogen-free

❖ KEY FEATURES

The APS-35V 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 12-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 APS-35V (Analog Type) provides the digital output data with the format of Analog Type. It can achieve ESD robustness, fast response time, high accuracy and linearity as well as long-term stability. All measure-ment data is fully calibrated and temperature compensated. In addition, it allows for easy system integration.

TYPICAL APPLICATIONS

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

❒ Maximum Ratings

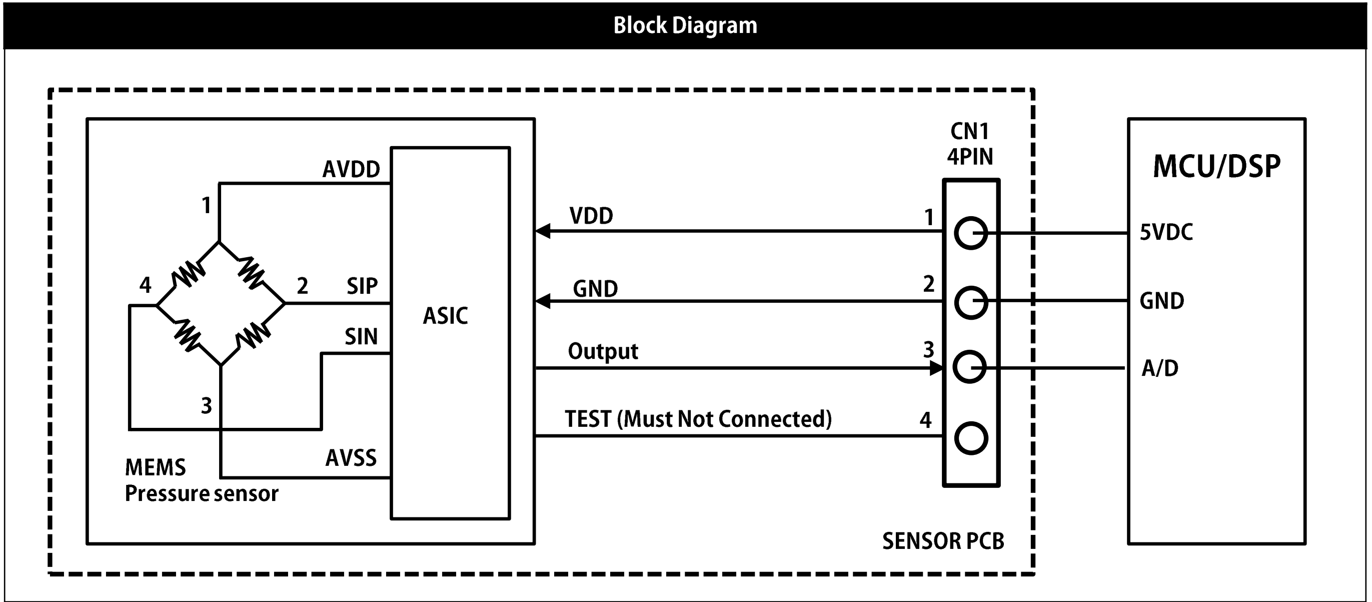
PARAMETER	MIN	TYP	MAX	UNITS
Analog Supply Voltage	3.0	-	6.0	V
Voltages at Digital and Analog I/O - In Pin	- 0.3	-	VDD+0.3	V
Voltages at Digital and Analog I/O - Out Pin	- 0.3	-	VDD+0.3	V
Storage Temperature Range (≥10 hours)	-40	-	85	° C
Storage Temperature Range (<10 hours)	-50	-	150	° C

❒ Recommended Operating Conditions

Specifications	
Pressure range	0 to 200···1,000Pa
Supply Voltage	3VDC or 5VDC
Supply Current	8.5mA
Output Range	0 to 90% of F.S Can be adjusted to application.
Output	Supply 3VDC - 0.5 to 2.5V Supply 5VDC - 1 to 3V
ADC resolution	Selection : 24bit
DAC resolution	Analog output : 16bit
ESD withstand	2KV
Humidity Limits	0 to 95% RH

Performance & Environmental	
Operating Temp	-20 to 80℃
Accuracy	±0.5% of F.S
Total error band	±1.0% of F.S(0 to 50℃) ±1.5% of F.S(-20 to 80℃)
Linearity	±0.2% of F.S
Cycle life	1,000,000 / cycle
Proof pressure	3 x F.S
Burst pressure	5 x F.S
Water protection	IP-60
External Material	PBT

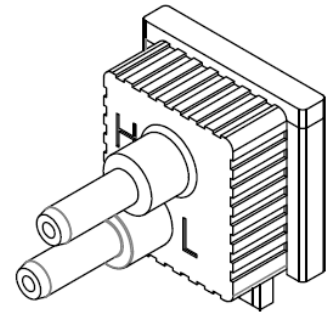
❒ Block Diagram

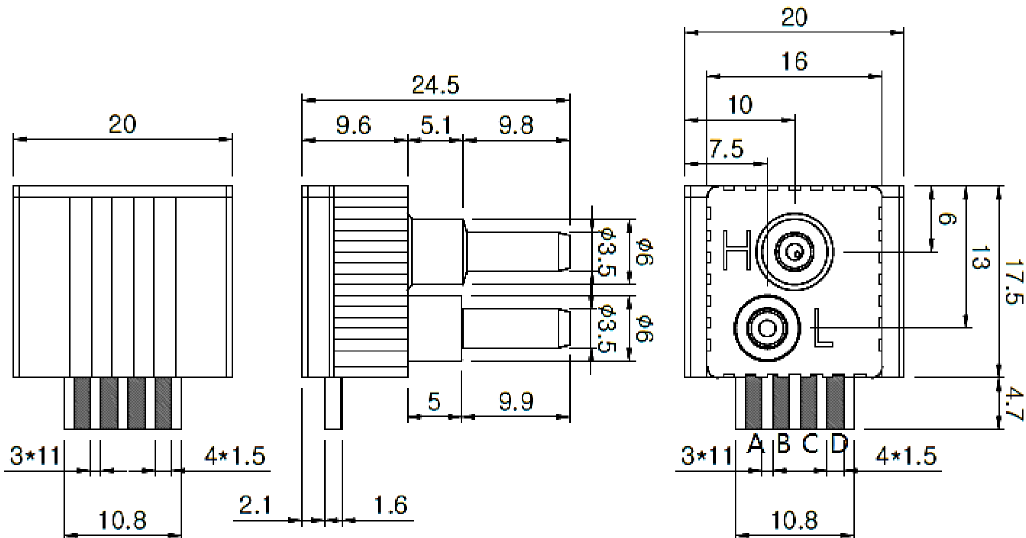


❒ Outline Drawing Dimension (mm)

Dimensions (mm)

PIN No.	Description	REMARK
1	VDD	3.0VDC or 5.0VDC
2	GND	GND
3	Output	Voltage Output
4	TEST	Must Not Connected





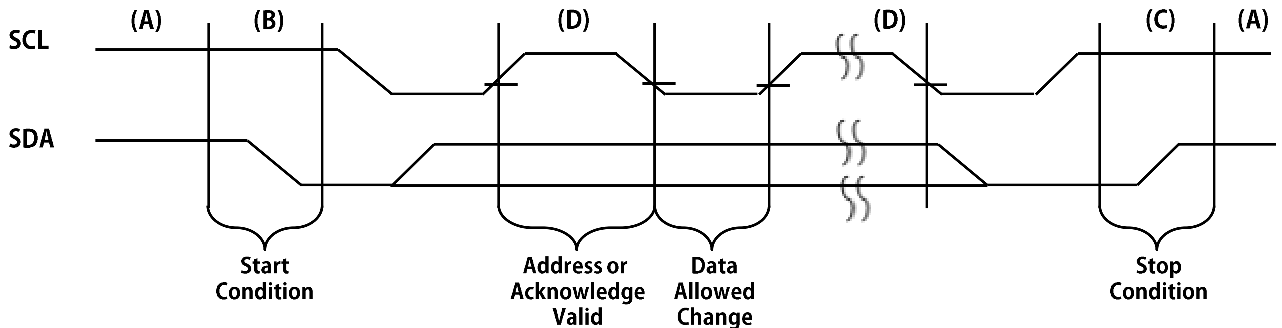
❒ I²C Operation

I²C Operation

The first command must be I²C command, the I²C mode will be selected. APS-35V 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, APS-35V 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 APS-35V 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 define.



❖ 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.

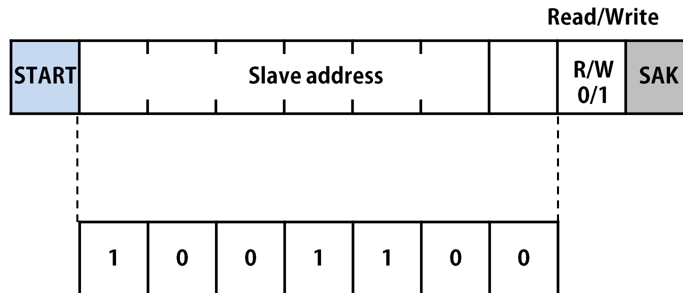
❖ Data valid as condition(D)
After a START condition, the data line is stable for the duration of the HIGH period of the clock signal. The data on the line must be changed during the LOW period of the clock signal. There is one clock pulse per bit of data. The number of valid data bytes transferred between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

❖ Acknowledge signal Each APS-35V 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 APS-35V 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.

❐ APS-35V control address

Chart

The seven bit is as slave address after START condition. The APS-35V 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			SAK		SAK	

Symbol	Meaning
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
~A	Master non-acknowledge
SAK	Slave acknowledge

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			SAK		SAK					SAK									

❐ I²C Command

I²C Command

The command of pressure measurement as shown below:

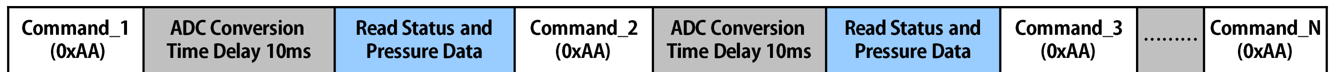
Command	Description
0xAA Hex (Force Mode)	(1) When the APS-35V is written a 0xAAHex, it will turn into force mode and start measurement pressure. (2) After pressure measurement is done, the APS-35V will turn into sleep mode automatically.

NOTE :

(1) Command Delay Time:

Before next command (0xAA) write to APS-35V , the APS-35V must has a delay time is more than 10 ms for ADC conversion time, as show below:

Command Delay Time Diagram



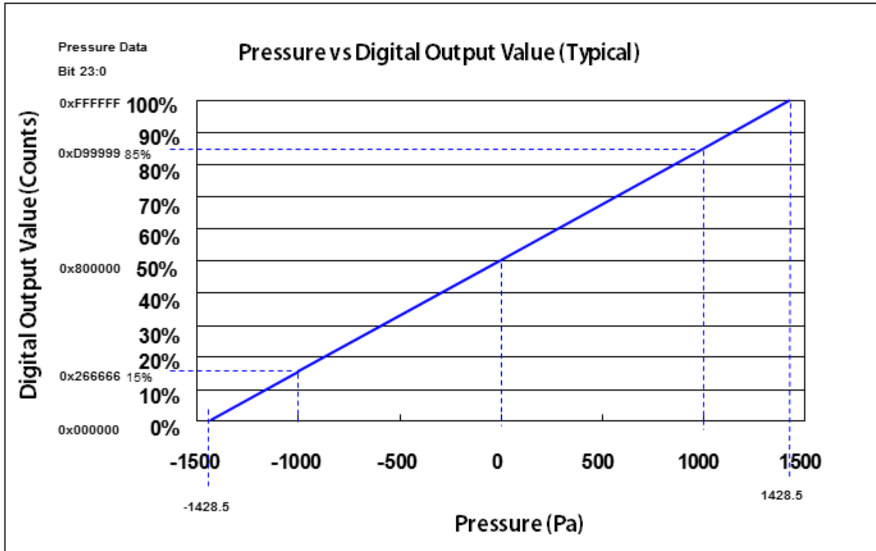
❐ Status Register

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 versus digital out value

chart

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



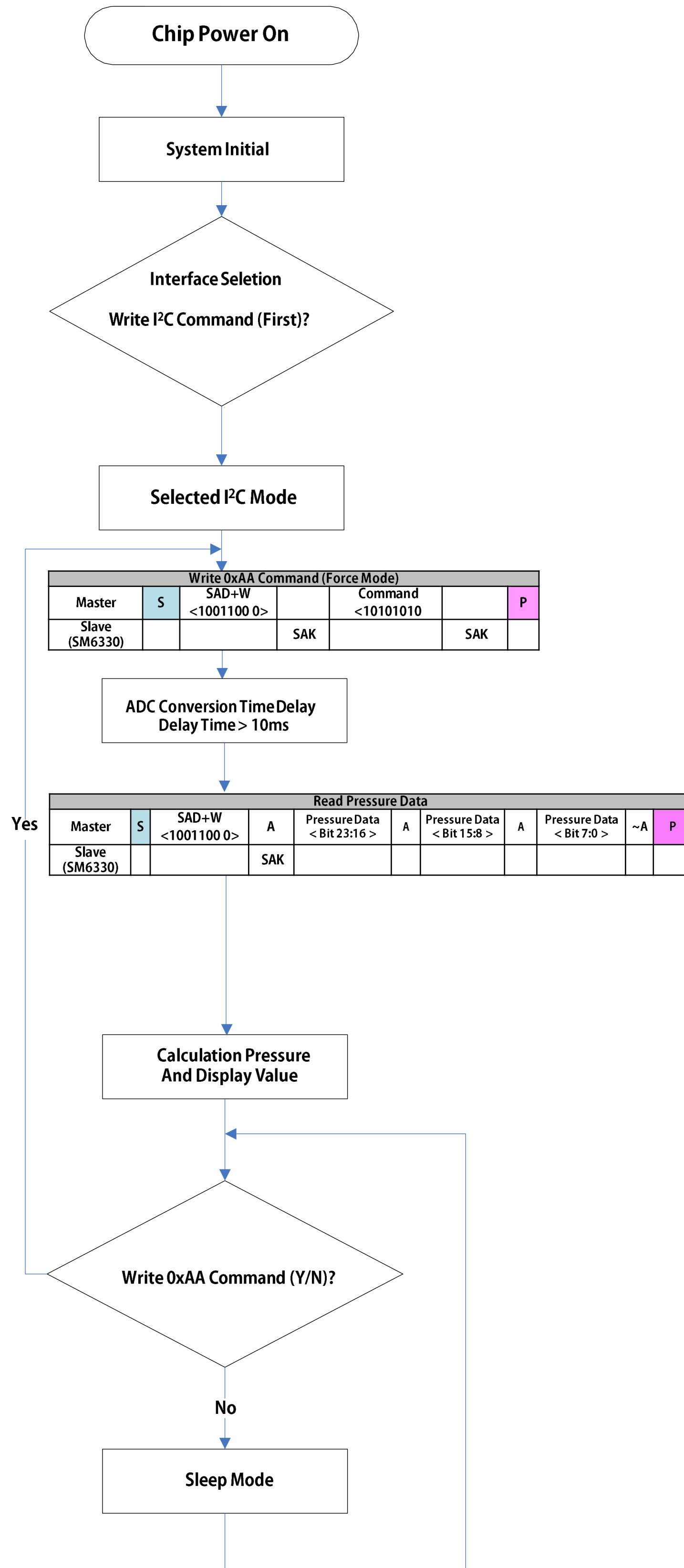
$$\text{Pout (Pa)} = \frac{(\text{ADC Value}_{(\text{Bit } 23:0)} - 0x800000_{\text{HEX}}) * (0x07d0_{\text{HEX}})}{(0xb33333_{\text{HEX}})}$$

❒ How to Order

Order Code INFO.

APS-35V	□	□	□
1) Pressure range			ǀ
0 to 200 Pa	2	ǀ	ǀ
0 to 500 Pa	4	ǀ	ǀ
0 to 1,000 Pa	1	ǀ	ǀ
Other on request	Z	ǀ	ǀ
2) Output signal		ǀ	ǀ
0.5 to 2.5V	------------	V1	ǀ
1 to 3V	------------	V2	ǀ
3) Supply Voltage			ǀ
3 VDC	------------	------------	3V
5 VDC	------------	------------	5V

❒ Pressure Measurement Operating low (I²C Mode)



❒ I²C Reading Code Example

Code Example

```
//* 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

Code Example

```
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

Code Example

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

Code Example

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