1.0 General Description

1.1 Scope

This application note is limited to AVSensors digital series pressure sensors with 14bit resolution. This includes the following product series. MCT-4D, MCT-5D, MCT-6D, MCT-SM58D, MCT-SM9333, MCT-SM9534.

1.2 Advanced Sensor, Multi Chip Technology, Digital Series

Advanced Sensors Multi Chip Technology (MCT) incorporates the latest mixed signal ASIC (Application Specific Integrated Circuit) with a bonded silicon gage to provide a leading *Digital Output* design for Industrial Transducers. The MCT Series provides a 14bit digital pressure and 11 bit digital temperature output offered in SPI and I^2C protocols. The rugged design is compatible with a wide range of harsh media including refrigerants, compressed air, and hydraulic fluids. The design superior performance provides 1% Total Error across a wide temperature range of -20 to 85°C and overall error of less than 2.5% over -40 to 125C. The flexible design incorporates many process fitting and connector types making it the ideal choice for OEM customers.

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

C code example for I2C with Read_DF4 command MCT-4D-DS-3BI002WDP.

On power-up, PORTB is initialized to all inputs with the internal pull-ups turned off, the external pull-ups pull the SDA and SCL lines high and the PORTB output latch bits SCL and SDA are initialized to zero. Routines WriteSDA and WriteSCL toggle their respective data direction bit depending on the value of parameter “state”. When state is a “1” the port pin is configured as input (external pull-ups pull high). When state is a “0” the port pin is configured as an output and the latch drives the pin low. WriteSDA and WriteSCL are very simple routines that could be incorporated into their respective calling routines to further reduce the code size.

General Calling Sequence for the Routines

```
SendStartBit();          /*start*/
SendByte(byte);          /*send address or command MSB first*/
GetOneByte();            /*read one byte from serial stream */
SendStop();              /*stop*/
```

PORTB on the ATmega164P is used to communicate with MCT-4D transducer. Bit assignments are as follows:

I2C.c

```
/*PB0 = SDA*/
/*PB1 = SCL*/

#include “i2c.h”

void WriteSCL(unsigned char state)
{
if (state)
DDRB &= 0xfd;            /* input ... pullup will pull high or Slave will drive low */
else
DDRB |= 0x02;            /* output ... port latch will drive low */
}
void WriteSDA(unsigned char state)
{
if (state)
DDRB &= 0xfe;            /* input ... pullup will pull high or Slave will drive low */
else
DDRB |= 0x01;            /* output ... port latch will drive low */
}
unsigned char SetSCLHigh(void)
{
WriteSCL(1);             /* release SCL*/
/* set up timer counter 0 for timeout */
t0_timed_out = FALSE;    /* will be set after approximately 34 us */
TCNT0 = 0;               /* clear counter */
TCCR0 = 1;               /* ck/1 .. enable counting */
/* wait till SCL goes to a 1 */
while (! (PINB & 0x02) && !t0_timed_out)
```


```
;
TCCR0 = 0;                          /* stop the counter clock */
return(t0_timed_out);
}
void BitDelay(void)
{
char delay;
delay = 0x03;
do
{
_NOP();
} while (--delay);
}
/* Routine SendStopBit generates an TWI stop bit assumes SCL is low stop bit is a 0 to 1
transition on SDA while SCL is high

              _____________
            /
SCL ___/

                        _____
                      /
SDA _________/
*/


void SendStopBit(void)
{
WriteSDA(0);
BitDelay();
SetSCLHigh();
BitDelay();
WriteSDA(1);
BitDelay();
}

/* Routine SendStartBit generates an start bit start bit is a 1 to 0 transition on SDA while SCL is
high

              _____________
            /
SCL ___/
```

```
        _________
                 \
SDA               \_____
*/

void SendStartBit(void)
{
WriteSDA(1);
BitDelay();
SetSCLHigh();
BitDelay();
WriteSDA(0);
BitDelay();
WriteSCL(0);
BitDelay();
}
unsigned char SendByte(unsigned char byte)
{
unsigned char i;
unsigned char error;
for (i = 0; i < 8; i++)
{
WriteSDA(byte & 0x80);   /* if > 0 SDA will be a 1 */
byte = byte << 1;              /* send each bit */
BitDelay();
SetSCLHigh();
BitDelay();
WriteSCL(0);
BitDelay();
}
/* now for an ack */
/* Master generates clock pulse for ACK */
WriteSDA(1);           /* release SDA ... listen for ACK */
BitDelay();
SetSCLHigh();          /* ACK should be stable ... data not allowed to change when
SCL is high */
/* SDA at 0 ?*/
error = (PINB & 0x01);     /* ack didn't happen if bit 0 = 1 */
WriteSCL(0);
BitDelay();
```

```
return(error);
}
unsigned char GetOneByte(unsigned char lastbyte)
{
/* lastbyte ==1 for last byte */
unsingned char i;
unsigned char data;
DDRB &=0xfe;                /* release SDA ... listen for slave output */
data=0;
for (i=0; i<8;i++)
{
SetSCLHigh();               /* Slave output should be stable ... data not allowed to
change when SCL is high */
BitDelay();
data=data<<1;
if (PINB & 0x01)
data=data | 1;
WriteSCL(0);
BitDelay();
}
/*send ACK*/
WriteSDA (lastbyte);            /* no ack on last byte ... lastbyte = 1 for the last byte */
BitDelay();
SetSCLHigh();
BitDelay();
WriteSCL(0);
BitDelay();
WriteSDA(1);
BitDelay();
return (data);
}
```

ReadWithPollingI2C.c

```
/*
ReadWithPollingI2C.c reads the digital output simply at any time and be assured the data is no
older than the selected response time specification by checking the status of the 2 MSBs of the
bridge high byte data
*/
#include “i2c.h”
extern unsigned char GetOneByte(unsigned char lastbyte);
extern unsigned char SendByte(unsigned char byte);
```

```
extern void SendStartBit(void);
extern void SendStopBit(void);
extern void BitDelay(void);
extern unsigned char SetSCLHigh(void);
extern void WriteSDA(unsigned char state);
extern void WriteSCL(unsigned char state);

unsigned char MCT-4D_Address;
unsigned char bufptr[4];
void Init (void)
{
__disable_interrupt();
/* P0 = SDA - bidirectional */
/* P1 = SCL - output */
/* P7, P6, P5, P4, P3, P2, P1, P0 */
/* O O O O O O O O */
/* 1 1 1 1 1 1 1 1 */
DDRB = 0xff;
PORTB = 0xfc;
/*setup MCT-4D device address*/
MCT-4DDO_Address=0x28;
/*
The factory setting for I2C slave address is 0x28, 0x36 or 0x46 depending on the interface type selected from the
ordering information. For this sample code, 0x28 is used for Slave address of MCT-4D.
*/
}
unsigned char ReadMCT-4D(unsigned char DF_Command)
{
unsigned char i;
unsigned char error;
SendStartBit();
if (SendByte((MCT-4D_Address<<1) + read))          /*send salve address byte*/
{
return (1);                                                    /*check error*/
}
for (i=0; i< (DF_Command-1); i++)
{
bufptr[i] = GetOneByte (0);                          /* 1 byte of read sequence */
}
bufptr[DF_Command-1] = GetOneByte (1);               /* 1 signals last byte of read sequence */
```

```
SendStopBit ();
return (0);
}
void main (void)
{
float Pressure, Temperature;
unsigned int Dpressure, Dtemperature;
float P1=819.15; /* P1= 5% * 16383 – B type*/
float P2=15563.85; /* P2= 95% * 16383 – B type*/
float Pmax=2.0;
float Pmin=-2.0;
Init();
do
{
ReadMCT-4D (DF4);                                    /*Read_DF4 command – data fetch 4 bytes */
If ((bufptr [0] & 0xc0) ==0x00)                      /*test status of the 2 MSBs of the bridge high byte of data*/
{
Dpressure= ((unsigned int) (bufptr [0] & 0x3f) <<8) + (bufptr [1]);
Dtemperature= (((unsigned int) bufptr [2]) <<3) + bufptr [3];
Pressure= (((float) Dpressure)-P1) * (Pmax-Pmin) / P2+Pmin;
Temperature= ((float) Dtemperature) * 200 / 2047 -50;
}
} while (1);
} /* main */

I2C.h
#include “iom164p.h”
#define DF2 2
#define DF3 3
#define DF4 4
#define write 0
#define read 1
```

2.3 SPI Code Example

C code example for SPI with Read_DF4 commandMCT-4D-DS-3BS002WDP）

ReadWithSPI.c

```
/*
ReadWithSPI.c reads the digital output simply at any time and be assured the data is no older than the selected
response time specification by checking the status of the 2 MSBs of the bridge high byte data */
/*PB0 = SCLK*/
/*PB1 = MISO*/
/*PB2 = SS*/
#include “iom164p.h”
#define DF2 2
#define DF3 3
#define DF4 4
unsigned char bufptr[4];
void Init(void)
{
/* P0 = SCLK – output */
/* P1 = MISO – input */
/* P2 = SS – output */
/* P7, P6, P5, P4, P3, P2, P1, P0 */
/* O O O O O O I O */
/* 1 1 1 1 1 1 1 1 */
DDRB = 0xfd;
PORTB = 0xfc;
}
void BitDelay(void)
{
char delay;
delay = 0x03;
do
{
while(--delay)
;
_NOP();
return;
}
unsigned char GetOneByte (void)
{
unsigned char data=0;
```

```
unsigned char i;
for (i=0; i<8; i++)
{
BitDelay();
SCLK=1;
BitDelay();
data=data<<1;
if (PINB & 0x02)
data=data | 1;
SCLK=0;
BitDelay()
}
return (data);
}
unsigned char ReadMCT-4D(unsigned char DF_Command)
{
unsingned char i;
SCLk=0;
SS=0;
BitDelay();
for (i=0; i<(DF_Command); i++)
{
bufptr[i] = GetOneByte ();                    /* 1 byte of read sequence */
}
SS=1;
BitDelay();
}
void main (void)
{
float Pressure, Temperature;
unsigned int Dpressure,Dtemperature;
float P1= 819.15;                             /* P1= 5% * 16383 – B type*/
float P2= 15563.85;                           /* P2= 95% * 16383 – B type*/
float Pmax= 2.0;
float Pmin= -2.0;
Init();
do
{
ReadMCT-4D (DF4);                             /*Read_DF4 command – data fetch 4 bytes */
If((bufptr [0] & 0xc0)==0)                    /*test status of the 2 MSBs of the bridge high byte of data*/
```

```
{
Dpressure= ((unsigned int) (bufptr [0] & 0x3f) <<8) + (bufptr [1]);
Dtemperature= (((unsigned int) bufptr [2]) <<3) + bufptr [3];
Pressure= (((float) Dpressure)-P1) * (Pmax-Pmin) / P2+Pmin;
Temperature= ((float) Dtemperature) * 200 / 2047-50;
```