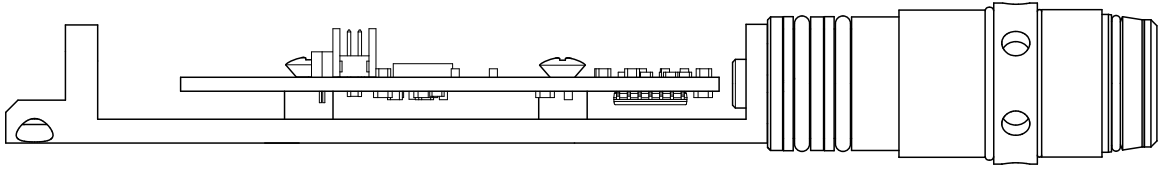
XtalX DDQS1 Quartz Pressure Transducer

Mechanical User Manual



www.phasesensors.com

XtalX DDSQ1 Quartz Pressure Transducer

Mechanical User Manual

4-016 National Institute for Nanotechnology
11421 Saskatchewan Drive
Edmonton, AB, T6G 2M9
TELEPHONE: (780) 264-2659

April , 2023

CONTENTS

SECTION 1
MECHANICAL INSTRUCTIONS

1.1 PROCESS FLUID FITTING

When removing or attaching the microcap to the pressure sensor, follow the procedures below:

Removal

1. Use a 11/16” hex wrench on the process end fitting and a spanner wrench (McMaster PN: 5472A1)
2. Turn the 11/16” wrench counter-clockwise to loosen.
3. If possible avoid scraping angled metal surface and PTFE O-ring groove with metal parts or process fitting threads.
4. Take care to ensure the you do not touch the membrane as it is fragile.
5. If cleaning is necessary, use Isopropanol/ acetone or other solvents to rinse the membrane without contacting it.
6. Replace PTFE O-ring after process end fitting is removed without scratching gland surfaces.

1.2 CONSUMABLES

The following O-rings may need to be replaced during the life of the sensor:

Number	O-Rings	Size	Qty.	Material
1 & 3	Electronics Housing Back-Up Ring	DN-014	2	PEEK
2 & 4	E-H O-Ring	DN-014	2	FFKM
5	Process-Side O-Ring	M14 x 1mm	1	PTFE

***Note:** The O-rings in the above table are not compatible with acetone.

The placement of the O-ring is shown below:

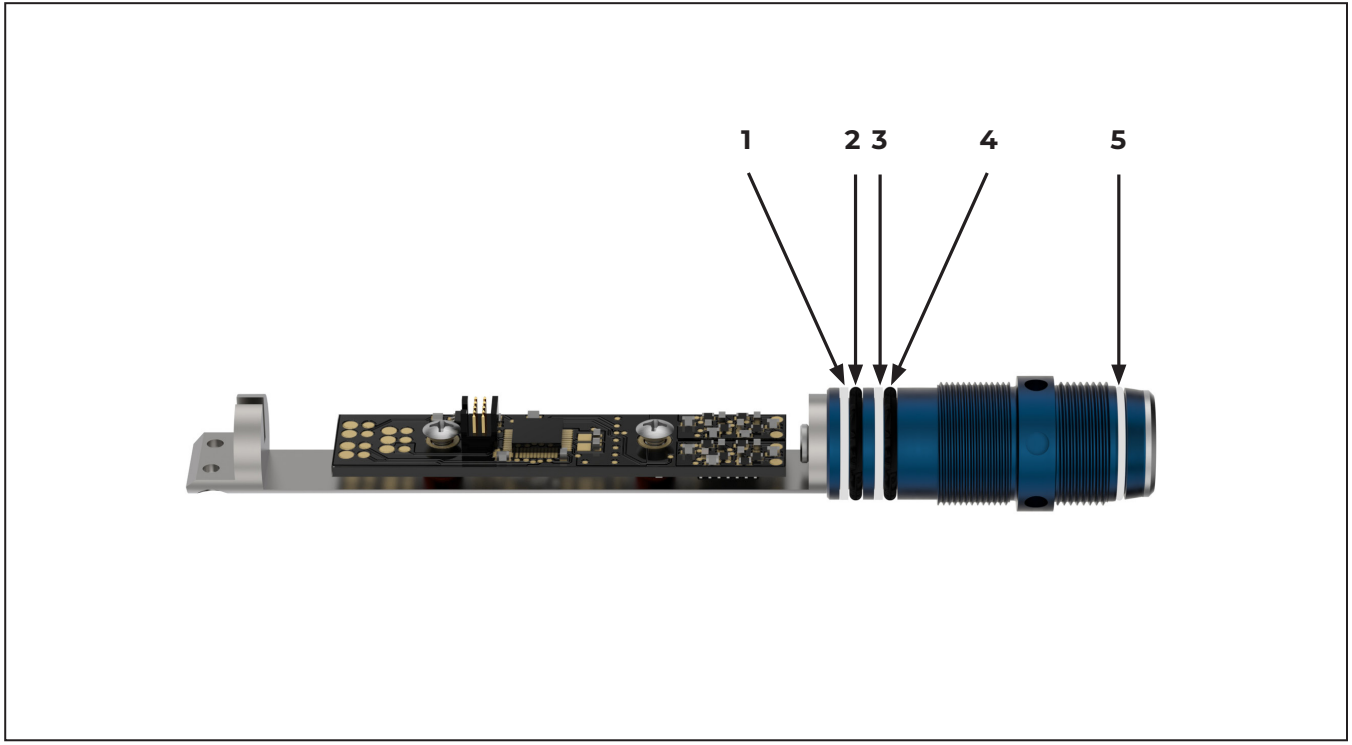


Figure 1-1. Placement of O-rings

1.3 MICROCAP ATTATCHMENT

1. Heat PTFE O-ring to 100°C for easier installation of O-ring.
2. Take care to ensure the you do not touch the membrane as it is fragile.
3. Engage the threads by hand while gently pushing the process end fitting.
4. Use a 11/16” hex wrench on the microcap and a spanner wrench on the feedthrough.
5. Turn the 11/16” wrench clockwise. Hand tighten process fitting to feedthrough end.
6. Torque to 40 N·m, do not over torque as this may damage metal to metal seal.
7. Gap between lip of process fitting and feedthrough should be approximately 0.02”.

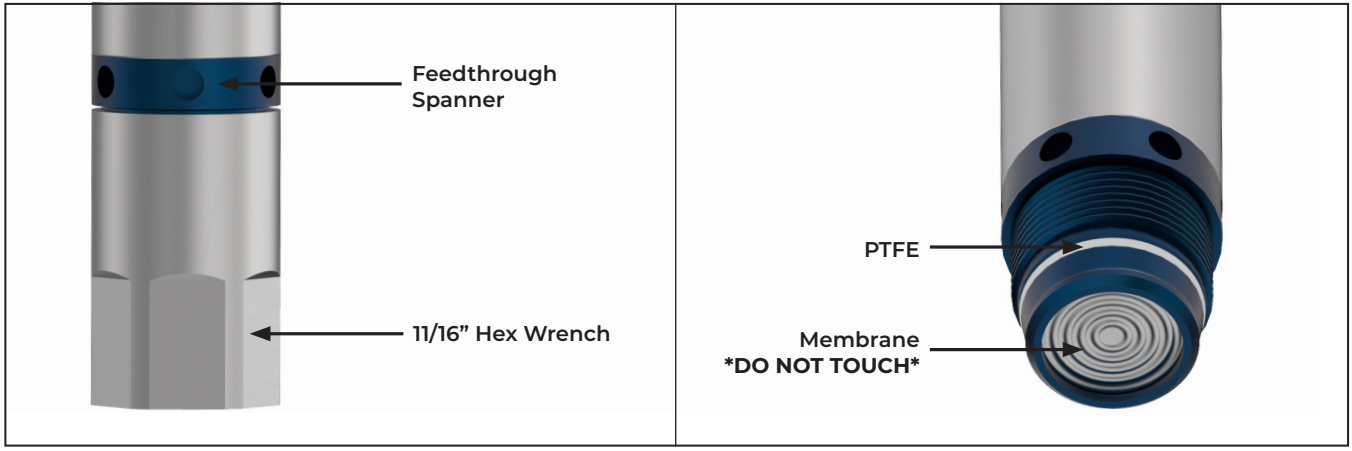


Figure 1-2. Microcap Attatchment

SECTION 2
XTALX DDQS1 SPECIFICATION SHEET

2.1 SPECIFICATION HIGHLIGHTS

CALIBRATED RANGE ..... ATM to 10, 000 PSI (1 to 690 BAR)
CALIBRATED TEMPERATURE RANGE ..... 20° TO 150°C
PRESSURE ACCURACY ..... 0.02 %F.S.
DRIFT @ MAX TEMPERATURE & PRESSURE 0.1%F.S./YEAR
TEMPERATURE ACCURACY ..... 0.01°C
DRIFT @ MAX TEMPERATURE & PRESSURE ..... 0.01°C/YEAR

2.2 MECHANICAL SPECIFICATIONS

WEIGHT: 325 GRAMS
HEIGHT: 152 mm (6”)
MAXIMUM WIDTH: 25 mm (0.75”)
PROOF PRESSURE: 103MPA (15,000PSI)
HOUSING MATERIAL: INCONEL 718

2.3 ELECTRICAL SPECIFICATIONS

MAXIMUM VOLTAGE RATINGS ..... -0.3 TO 13.5V
DC VOLTAGE SUPPLY RANGE ..... 2.9 TO 12V
DC CURRENT DRAW @ 25°C ..... 29.2 mA
CURRENT DRAW @ F.S. TEMP ..... 37.5 mA
OUTPUT TTL-232R-2V7
BAUD RATE 57600
ESD ..... ±2kV
STARTUP TIME & SETTLE TIME ..... 230 ms
MAX CURRENT CONSUMPTION ..... 40 mA
LOW POWER STATE CURRENT DRAW 1s SAMPLE @ 25°C ..... 1.212 mA
LOW POWER STATE CURRENT DRAW 1s SAMPLE @ F.S TEMP.. 2.9 mA
LOW POWER STATE CURRENT DRAW 9s SAMPLE @ 25°C ..... 400 uA
LOW POWER STATE CURRENT DRAW 9s SAMPLE @ F.S TEMP 700uA

2.4 ADDITIONAL SPECIFICATIONS

STORAGE TEMPERATURE ..... -65° to 150°C
ACHIEVABLE RESOLUTION 0.002psi
REPEATABILITY ..... 0.05% F.S.
NOMINAL SENSITIVITY 3psi/Hz
RESPONSE TIME 0.1s
GRAVITATIONAL EFFECTS ..... NEGLIGIBLE
ORIENTATIONAL EFFECTS ..... NEGLIGIBLE
ACCELERATION SENSITIVITY ..... NEGLIGIBLE
STARTUP TIME @ 25°C ..... 130 ms
PEAK INRUSH CURRENT @ 25°C ..... 37.12 mA
STARTUP TIME @ F.S. TEMP ..... 150 ms
PEAK INRUSH CURRENT @ F.S. TEM ..... 40 mA
AVERAGE LIFETIME EXPECTANCY @ 150°C ..... 2 YEARS
AVERAGE LIFETIME EXPECTANCY @ 175°C ..... 6 MONTHS
AVERAGE LIFETIME EXPECTANCY @ 210°C ..... 30 DAYS

2.5 LIFE TIME EXPECTANTCY NOTE:

Operating at extreme temperatures can and will dramatically reduce life expectancy. The expected lifetimes above were observed in a dry environment where both the electronics and pressure fittings were subject to atmosphere. XS-HTI-4 is not rated for operation beyond 175°C and 10000psi; accuracy and quality will not be guaranteed.

2.6 FREQUENCY RESPONSE NOTE:

An increase in pressure applied to the sensor will result in a decrease of the pressure oscillator frequency (approximately 3 PSI/Hz). Transient temperature effects will influence the pressure oscillator frequency. Therefore, the temperature oscillator is used in tandem. An increase in temperature of the sensor will result in an increase of the temperature oscillator frequency (approximately 0.1°C/Hz). Pressure readings during temperature stability will result in optimal performance.

2.7 DDQS1-10000-150 SPECIFICATION DRAWINGS

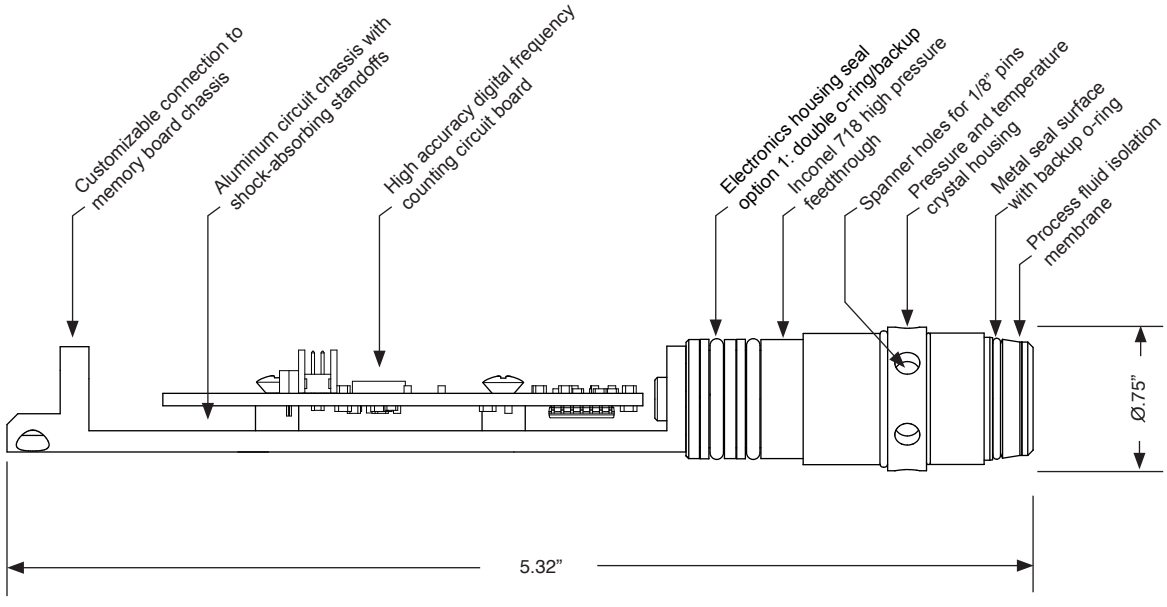


Figure 2-1. DDQS1 Pressure Transducer Specification Drawing

SECTION 3
COMMUNICATIONS SET UP

3.1 OVERVIEW OF COMMUNICATIONS ARCHITECTURE

Power, Ground, Rx, Tx, Clkin, and ClkCal are the 6 connections available to be made between the pressure transducer and the memory module.

Power input requires a minimum voltage of 2.2V DC and will accept up to 10V DC, with an absolute maximum ratin of 13.5V DC. It is recommended to minimize the DC voltage supplied to the transducer as it will consume the same amount of current regardless of voltage supplied in the appropriate range.

Please note that the communications returned by the transducer's level shifter will have a logic level of the power supplied to the board.

I.e. If the memory module or serial cable attached to the circuit board is providing 3.3V DC on the power pin, then the serial communication returned will have a logic high voltage of 3.3V.

Only the ClkIn signal, Rx, and Tx incorporate this level shifter.

Communications Architecture on this transducer comprises of three bi-irectional level shifters for it's serial communication and an additional data line pin.
The utlization of having an independent MOS-FET connecting each data line allows both the transducer and memory module designer to optimize their own power consumption architecture.

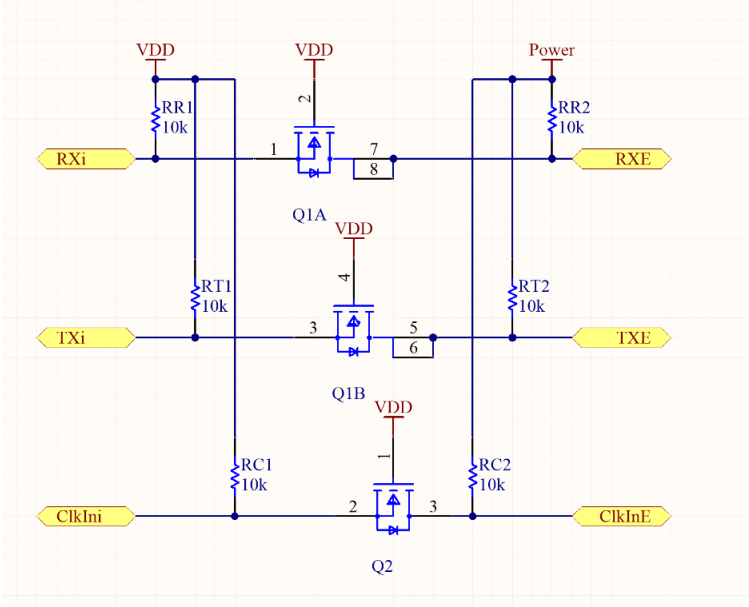


Figure 3-1: Communication Level Shifter Schematic

The MOS-FETs can operate in three states:

1. Neither side is pulled low and because the voltage difference between the Gate (pin 2 on Q1A in Fig 3-1.) and the Source (pin 1 on Q1A in Fig 3-1.) is below the threshold voltage of the MOS-FET the device does not conduct. Neither side is transmitting data.
2. Data is being sent by the Transducer and it's microcontroller drives the Source (pin 1 on Q1A in Fig 3-1.) low during communication. When the Source Pin is driven low, the voltage difference between the Gate (pin 2 on Q1A in Fig 3-1.) and the source pin becomes equal to VDD (the operating voltage of the Transducer) and if VDD is greater than the threshold voltage of the MOS-FET the MOS-FET conducts which then drives the Drain (pins 7 and 8 on Q1A) low.
3. Data is being sent by the Memory module, or other serial line, and the Drain (pins 5 and 6 of Q1B) is driven low during communication. The drain-substrate diode of the MOS-FET begins to pull the source lower until the voltage difference between the Gate and Source is greater than the threshold voltage of the MOS-FET and the MOS-FET conducts, making the Source low.

A third MOS-FET called CLKIn is included in this architecture for any additional data lines needed sourced by the micrcontroller. Often this pin can be used to signal that the data is about to be transmitted when in autonomous mode.
Lastly a final pin, CLKCal, is available which connects directly to the micrcontroller without a MOSFET. This pin can also be used as an additional data line. Precautions should be made to avoid exposing this pin to voltages greater than 3.3V DC.

3.2 SOFTWARE AND COMMUNICATIONS SETUP

Operating conditions for the interface:

Signal	PCB Symbol	Min	Typ	Max
GND	-	0V	0V	
PWR	+	2.2V (1)	3.0V	12V
TXD	TX	0V		PWR
RXD	RX	0V		PWR
Wakeup	CLKCal	0V		5V (2)
Vhigh		2.2V		
Vlow				2.2V

(1) PWR is recommended to be >= 2.4V.

(2) CLKCal higher than 5V will permanently damage the transducer.

The transducer includes TX/RX level-shifters that will shift the serial signal up/down to the PWR level provided by the external memory board (the transducer's internal microcontroller operates at a lower voltage). When transmitting from the memory board to the transducer, any voltage below 2.2V is interpreted as a logic low and any voltage above 2.2V is interpreted as a logic high. When transmitting from the transducer to the memory board, a logic low will be transmitted as 0V and a logic high will be transmitted as PWR. The TX/RX nomenclature is from the perspective of the

transducer: the transducer transmits on the TX pin and receives on the RX pin.

To communicate with the XtalX pressure transducer:

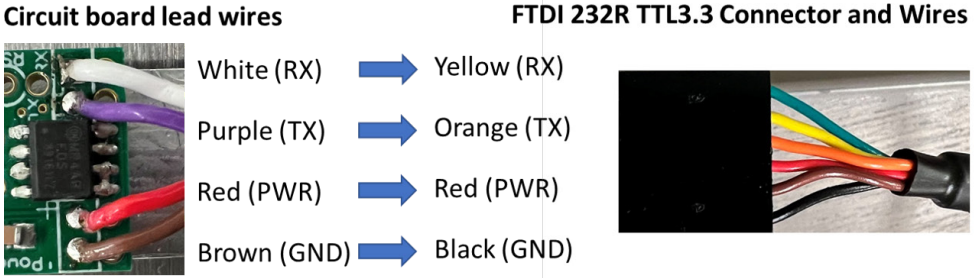


Figure 3-2: Connection to an FTDI TTL-232R-3V3 Cable

1. With the provided TTL-232R-3V3 FTDI cable disconnected from the computer, connect sensor's DIP pins to the FTDI cable, aligning the brown wire from the sensor with the black wire on the cable. If the sensor has no DIP header attached, consult the following wiring diagram to solder your cable directly to the sensor:

2. Make a note of which serial devices are currently available on the computer so that we can identify the newly-connected FTDI cable afterwards.
   - On Windows: Open the Device Manager and list all the available COM ports.

- On macOS: Open a Terminal window and type the following: ls -l /dev/cu*serial*

3. Connect the FTDI cable to your computer's USB port, and then make note of the newly-added port:
   - On Windows: The Device Manager window will automatically refresh itself.
   - On macOS: In the same Terminal window as above, repeat the "ls -l" command.

   The newly-added port is the new entry in either the Device Manager window or the "ls -l" output.

4. To communicate with the sensor, the serial port should be configured for 57600 baud, no stop bits, 8 data bits, 1 parity bit (57600 N81). On Windows, you can use your favorite serial terminal emulator such as Putty or Realterm. On macOS, in a terminal window you can use the screen command to directly connect by typing:

   screen /dev/cu.usbserial-XXXXXXX 57600.

   The /dev path should be replaced with the path noted in step 3. To exit the screen tool, type "Ctrl-A" followed by "k" and answer "Y" to the exit prompt.

5. When power is applied to the sensor, it outputs some logging information as it boots up and then enters command mode:

```
R: xhtifw xhti7 2.1.6
r: Software Reset
G: 29093b43a062702a806df5161164f3efbbead328
c: gcc 9.3.1 20200408 (release)
I: XHTI-9-1000166 2022-10-12T19:05:33
S: Starting LS/HS clocks...
S: Waiting for HS clock to finish starting...
H: HS clock is hardcoded to 21 MHz.
P: Max PLL @ 168000000 Hz (M=2, N=32, R=2)
S: Waiting for LS clock to finish starting...
S: Waiting for LS clock to stabilize...
S: LS/HS = 1024/656260
S: LS clock is 32767.50 Hz.
S: Initialization complete.
=
```

If you don't see any sensor output, it may have completed the boot sequence before your terminal emulator was started. You can try issuing a reset command by typing a capital "R" followed by the

Enter key to manually reset the sensor.

6. If power-on testing was successful, the sensor boot sequence will end with a “=” line, indicating it is ready to accept a command. The AUT command, for instance, can be used to place the sensor into autonomous mode and periodically transmit a measurement. Type “AUT3” followed by Enter to place it in a mode where it will transmit a new measurement every 3 seconds (note that characters are not echoed when typing commands):

```
...
S: LS clock is 32767.55 Hz.
S: Initialization complete.
=
A: Starting autonomous mode.
M: TFFFFFFFF PFFFFFFFF
M: TFFFFFFFF PFFFFFFFF
M: TFFFFFFFF PFFFFFFFF
M: TFFFFFFFF PFFFFFFFF
...
```

In this example, the “FFFFFFFF” values would be replaced by real temperature and pressure transducer counts. Consult the Command Reference section for other commands and their

responses.

7. A fifth pin, labeled “CLKCal” on the sensor, can be used to facilitate memory board interaction. This pin will be driven low by the transducer approximately 20 ms before transmitting a measurement over the serial lines, allowing the memory board to wake up and do any pre-processing necessary to accept a measurement. This pin will continue to be held low for the 100ms measurement period and then will finally be released so that an external pull-up can bring it back high. If the memory board doesn’t need any special signaling, the CLKCal pin can be a no-connect. Unlike the other pins, this pin has an absolute maximum voltage of 5V, so the external pullup should take care not to exceed this limit.

8. A “break” condition on the serial lines can be used to reset the sensor. A serial break condition is defined as holding the data line low for longer than a start bit and 8 data bits. In order to actually reset the sensor, the break condition must be present while the sensor wakes up to perform a measurement. For instance, to reset a sensor that is performing a measurement every 5 seconds the break condition should be held for at least 5 seconds. Consult the documentation for your terminal emulator program to learn how to transmit a break condition.

9. If the memory board is able to control transducer power, it may also reset the transducer by removing power for at least 1 second and then restoring power, which will result in a cold boot.

4 SERIAL COMMAND REFERENCE

When the transducer boots up after power is applied or after a reset event occurs, it will print some startup logs on the serial port and then enter command mode. In command mode, the transducer waits for a command to be received on the RX serial pin. Commands are transmitted as a short sequence of ASCII characters and are terminated by a '\r' (0x0D) carriage-return character. Note that '\n' (0x0A) characters are ignored, so if your terminal emulator transmits a CRLF ("\r\n") sequence when you press Enter it should work fine but if it only transmits a '\n' character then commands will not work properly. The transducer does not echo characters back to the memory board.

Upon receiving a command, the transducer will process it and typically then transmit a response. Most responses are single-line and are terminated by a "\r\n" sequence, however some responses span multiple lines and are terminated by a "\r\n=\r\n" sequence. These commands are listed explicitly in their descriptions. Responses typically start with an "S: " or an "E: " prefix indicating successful completion or an error. Commands are listed in alphabetical order, below.

4.1. AUT<N>

The AUT command places the transducer into autonomous mode. The <N> parameter is an integer in the range 1 to 86400, inclusive, indicating the period in seconds between the start of consecutive measurements. The transducer enters a low-power sleep mode between measurements and

once autonomous mode has been started the transducer will no longer accept commands. To exit autonomous mode and return to command mode, a sensor reset using a serial break condition or a power cycle is required.

The transducer will transmit measurements in ASCII format representing the number of counts for the temperature and pressure crystals. After receiving an AUT command, the transducer will immediately acknowledge receipt of the command by printing:

```
A: Starting autonomous mode.
```

It will then perform a measurement and immediately go to low-power sleep mode. After N seconds elapse, the sensor will wake up, transmit the initial measurement and start a new measurement. The output looks like this:

```
M: T01004C69 P00FC30C3
M: T01004C65 P00FC309D
M: T01004C61 P00FC309F
...
```

Measurements lines start with the “M: “ prefix and then include a count for the temperature and pressure crystals, respectively. A new measurement line will be emitted every N seconds and

transmits the result of the previous measurement. If the memory board is using the CLKCal wakeup signal, it will be asserted (driven low) around 20 ms before the measurement line is transmitted.

The temperature and pressure counts can be converted to frequencies for use with the sensor polynomials as follows:

```
double Ft = PLLClk * 26200 / t_count;
double Fp = PLLClk * 5000 / p_count;
```

The PLLClk value is a per-sensor constant defining the frequency of the reference used for counting and can be retrieved using the HDR command.

4.2. aut<N>

The aut command is identical to the AUT command except that the output will be in a binary format instead of ASCII to minimize the transmission time and battery impact and includes a CRC to ensure correct transmission of data. Consult the AUT command for details on how autonomous mode works.

In aut mode, measurements are transmitted using the following 10-byte packed binary format instead of using ASCII:

```
struct tp_measurement
{
   uint8_t    hdr[2];
   uint8_t    iteration;
   uint8_t    t_count[3];
   uint8_t    p_count[3];
   uint8_t    crc8;
};
```

The transmitted data begins with a two-byte header, 0x00 0x55. This sequence can be used by some microcontrollers such as a PIC to wake up and automatically set the baud rate of the UART. Following the header is an 8-bit iteration number that starts at 0 and increments after every measurement, wrapping back to 0 when it hits 256. Following the iteration number are two 24-bit numbers representing the temperature and pressure counts. These measurements are stored in little-endian order (least-significant byte first) and can be converted to frequencies for use with the sensor polynomials as follows:

```
uint32_t t_count = ((tpm.t_count[0] <<  0) |
                    (tpm.t_count[1] <<  8) |
                    (tpm.t_count[2] << 16)) + BIAS_COUNT;
uint32_t p_count = ((tpm.p_count[0] <<  0) |
                    (tpm.p_count[1] <<  8) |
                    (tpm.p_count[2] << 16)) + BIAS_COUNT;
double Ft = PLLClk * 26200 / t_count;
double Fp = PLLClk * 5000 / p_count;
```

The BIAS_COUNT value is a per-sensor constant that is used to help compress the measurement into a 24-bit value and the PLLClk value is a per-sensor constant defining the frequency of the reference used for counting. They can both be retrieved using the HDR command.

Finally, an 8-bit CRC value over the first 9 bytes of the measurement (including header fields) is appended. Consult Appendix A for sample code to compute the CRC value for validation.

4.3. CAL

The CAL command puts the sensor into a calibration-like mode where it streams readings without entering sleep mode or delaying between measurements. The CAL command always uses ASCII

characters and includes counts for calibration clocks that are not included on the final sensor. The output for the CAL command is similar to the AUT command:

```
...
M: T00FD1DAA P00DC556A L00FFFE9E CFFFFFFFF
M: T01004ACE P00FC3785 L00FFFE9C CFFFFFFFF
M: T01004ACD P00FC3477 L00FFFE9B CFFFFFFFF
M: T01004ACD P00FC331A L00FFFE9C CFFFFFFFF
...
```

The T and P values are counts for the temperature and pressure crystals, while the L value is the count for the 32.768 kHz RTC crystal. The C value is unused. The CAL command is most useful to stream continuous measurements from a sensor that has a permanent power source rather than a battery. In calibration mode the sensor continues to respond to other commands, however if any other commands are required it is recommended to first reset the sensor using the “R” command, then issue the other commands and then finally return to calibration mode using a new CAL command.

4.4. ECH<string>

The ECH command echoes any characters back to the client for debugging or synchronization

purposes. It can be useful when controlling the sensor through a scripted interface to ensure that responses from the sensor are synchronized with the commands being sent from the script.

4.5. FLC

The FLC command clears all entries from the sensor's internal flash log. Prior to deploying the sensor at a measurement site it is recommended to clear the flash log so that as much backup data as possible can be stored and recovered in the event of a memory board failure.

4.6. FLD

The FLD command dumps the entries in the flash log to the serial port. This can potentially write hundreds or thousands of lines of measurements to the serial port. When the sensor is performing autonomous mode measurements, every 5 minutes a record is stored in the flash log that contains a summary of the minimum, maximum and average counts for each of the temperature and pressure crystals. In the event of a memory board failure, the transducer flash log can be recovered to yield a low-resolution record of the measurements.

Since the FLD command can yield many lines of response text, the end of the response is indicated by an "\r\n=\r\n" sequence.

4.7. FLt<N>

The FLt command inserts N test entries to the end of the flash log for debugging/testing purposes.

4.8. FLT<N>

The FLT command prints the most-recent N entries from the flash log. This is useful to look at new entries in a long flash log.

4.9. HDR

The HDR command prints information about the calibration header stored in the transducer flash. It has output similar to the following:

```
S: RefClk .0 Id 0 Bias 12053700 PStartupMs 800 PLLClk 168000000
=
```

The output is a key-value dictionary where the keys and values are separated by space characters. The RefClk value is a former calibration value that has been deprecated. The Id value identifies the most-recent calibration report for the sensor. The Bias value is a number that must be added to binary-mode autonomous measurements to transform the 24-bit temperature and pressure values

into actual counts for use with the pressure and temperature polynomials. The PStartupMs value indicates the length of time in milliseconds that it takes for this specific pressure crystal to stabilize from either a cold boot or from an autonomous-mode wakeup; the value differs from crystal to crystal. The PLLClk value specifies the reference clock frequency used for measuring crystal counts.

4.10. MP

The MP command prints the firmware manifest and can be used to detect which version of the transducer firmware and which version of the bootloader firmware are present on the sensor. It has output similar to:

```
1*: 0x0217 xhtifw
2 : 0x0099 bootloader_uart
=
```

4.11. PLP

The PLP command prints the pressure polynomial. The pressure polynomial is a bivariate polynomial that takes the pressure and temperature crystal frequencies as inputs and yields a temperature-compensated pressure measurement in absolute PSI as an output. The command output is similar to the following:

```
40E5C144C2ED8A82,40E8759C7137D262
410FFDB0FB68FCBA,41101818107E430E
40BFC283613F2CB0,C0745887C9B56710,C0501EA1E706FC72,C032026F11337F1E,C024B800E816D096
C0BEB32C0669832E,40686356A9CDDD20,40243504D9A88C2D,401A82F2FE88D984,40142FC2B6BFAC2A
C07424D2B56C1955,4044D0F3E5686845,403C4CCFF2501125,C03EF155EA0D6903,C040EA01791ABC6F
4015A4D839E52887,C02F798A7F87F900,C03E4A4AC7CA2414,C01816620051DD74,401FF4835EB95910
C019F54E39564540,C03EC3B67776D645,C047DA829AEE9658,40310897086006F7,4041414C8A2402D4
=
```

Each number is the hexadecimal representation of a 64-bit double-precision floating-point value. Consult Appendix B for more information on how to convert these values into usable numbers. The example above translates to the following set of floating-point values:

```
 44554.14879490902,50092.888820563225
262070.12275884097,263686.01610665105
8130.513202618036,-325.5331513486899,-64.47863174135435,-18.009507250854874,-10.359381916794785
-7859.171972841765,195.1043290158059,10.103552629303602,6.627880074598924,5.046641211937677
-322.30144254900125,41.63244311903012,28.300048012302017,-30.942717197679496,-33.82816995435211
5.410981087314332,-15.737384782169556,-30.290203558779538,-6.021858220097204,7.988782386839617
-6.489556213274511,-30.764502970230996,-47.7071107544877,17.033554576343672,34.51014830358841
```

The first line lists the normalization range for the pressure frequency. The pressure frequency input to the polynomial is a normalized value that linearly maps values in this normalization range to the range [-1, 1]. The normalization range is a per-calibration value and will be different for all sensors and even for different calibrations applied to the same sensor. Let P0 and P1 be the low and high ends of the normalization range. Then, a given pressure frequency, Fp, would be mapped using the following equation:

```
Fp_norm = 2 * (Fp - P0) / (P1 - P0) - 1
```

For instance, using the values from the example output above, for a measurement of Fp = 49000 Hz we would have the following:

```
    P0 = 44554.14879490902
    P1 = 50092.888820563225
Fp_norm = 2 * (Fp - P0) / (P1 - P0) - 1
      = 0.6053655468567911
```

The second line lists the normalization range for the temperature frequency. This works the same way as normalization for the pressure frequency; it linearly maps the frequency values in the normalization range to the range [-1, 1]. A given temperature frequency, Ft, would be mapped using the following equation:

```
Ft_norm = 2 * (Ft - T0) / (T1 - T0) - 1
```

For instance, using the values from the example output above, for a measurement of Ft = 262345 Hz we would have the following:

```
    T0 = 262070.12275884097
    T1 = 263686.01610665105
Ft_norm =  2 * (Ft - T0) / (T1 - T0) - 1
      = -0.6597829410814067
```

The rest of the output consists of the coefficients for each term of the bivariate polynomial. In the example above, this corresponds to a 5 x 5 matrix of coefficients. The pressure polynomial takes two inputs, Fp_norm and Ft_norm. For simplicity, we will call these inputs P and T, respectively. Each column of the matrix is a power of P from 0 to 4 and each row of the matrix is a power of T from 0 to 4.

So, cells in the 5 x 5 matrix correspond to the coefficients for each of the following terms:

```
1,    P,       P**2,       P**3,       P**4,
T,    P * T,    P**2 * T,    P**3 * T,    P**4 * T,
T**2, P * T**2, P**2 * T**2, P**3 * T**2, P**4 * T**2
T**3, P * T**3, P**2 * T**3, P**3 * T**3, P**4 * T**3
T**4, P * T**4, P**2 * T**4, P**3 * T**4, P**4 * T**4
```

Working through our example above, we would set:

```
P = 0.6053655468567911
T = -0.6597829410814067
```

And then we would compute the temperature-compensated pressure using the formula:

```
psi =  8130.513202618036000 * 1 +
       -325.533151348689900 * P +
        -64.478631741354350 * P**2 +
        -18.009507250854874 * P**3 +
        -10.359381916794785 * P**4 +
      -7859.171972841765000 *       T +
```

```
      195.104329015805900 * P    * T +
       10.103552629303602 * P**2 * T +
        6.627880074598924 * P**3 * T +
        5.046641211937677 * P**4 * T +
     -322.301442549001250 *       T**2 +
       41.632443119030120 * P    * T**2 +
       28.300048012302017 * P**2 * T**2 +
      -30.942717197679496 * P**3 * T**2 +
      -33.828169954352110 * P**4 * T**2 +
        5.410981087314332 *       T**3 +
      -15.737384782169556 * P    * T**3 +
      -30.290203558779538 * P**2 * T**3 +
       -6.021858220097204 * P**3 * T**3 +
        7.988782386839617 * P**4 * T**3 +
       -6.489556213274511 *       T**4 +
      -30.764502970230996 * P    * T**4 +
      -47.707110754487700 * P**2 * T**4 +
       17.033554576343672 * P**3 * T**4 +
       34.510148303588410 * P**4 * T**4
   = 12876.177498074392
```

So, a pressure frequency reading of 49000 Hz and a temperature frequency reading of 262345 Hz correspond with a temperature-compensated pressure value of approximately 12876.18 PSI.

4.12. PLT

The PLT command prints the temperature polynomial. The temperature polynomial is a univariate polynomial that takes the temperature crystal frequency as an input and yields a temperature measurement in degrees C as an output. The command output is similar to the following:

```
410FFDB0FB68FCBA,41101818107E430E
405944BB9EF01C55,4053A789F77C8F96,BFF1B601C1C69622,3FF6546EE9620121
=
```

Each number is the hexadecimal representation of a 64-bit double-precision floating-point value. Consult Appendix B for more information on how to convert these values into usable numbers. The example above translates to the following set of floating-point values:

```
262070.12275884097,263686.01610665105
101.07395146797474,78.61779582180165,-1.1069352692951573,1.395613586093596
```

The first line lists the normalization range for the temperature frequency. It linearly maps the frequency values in the normalization range to the range [-1, 1]. A given temperature frequency, Ft,

would be mapped using the following equation:

```
Ft_norm = 2 * (Ft - T0) / (T1 - T0) - 1
```

For instance, using the values from the example output above, for a measurement of Ft = 262345 Hz we would have the following:

```
    T0 = 262070.12275884097
    T1 = 263686.01610665105
Ft_norm =  2 * (Ft - T0) / (T1 - T0) - 1
      = -0.6597829410814067
```

The rest of the polynomial consists of the coefficients for each term of the polynomial in powers of T from 0 to 3. Working through our example:

```
T = Ft_norm = -0.6597829410814067
deg_c = 101.0739514679747400 +
        78.6177958218016500 * T +
        -1.1069352692951573 * T**2 +
         1.3956135860935960 * T**3
    = 48.32056943618824
```

So, a temperature frequency reading of 262345 Hz corresponds with a temperature of approximately 48.32C.

4.13. R

The R command causes the sensor to reset. Note that neither this command nor any other command is recognized when the sensor is in autonomous mode.

4.14. SER

The SER command prints the sensor serial number and the date that the sensor was commissioned (when a new PCB is flashed with initial firmware and assigned a serial number). Sample output:

```
XHTI-9-1000171
2023-03-28T19:22:13
=
```

Appendix A

==========

The binary data format for autonomous mode measurements includes a CRC value to check that the measurement was properly transmitted. This is helpful to discard corrupt readings in harsh communication environments. The following C code can be used to compute the CRC for validation:

```
#define POLYNOMIAL 0x9B

uint8_t crc_byte(uint32_t v, uint8_t P)
{
   v = (v & 0x80 ? (v << 1) ^ P : (v << 1));
   v = (v & 0x80 ? (v << 1) ^ P : (v << 1));
   v = (v & 0x80 ? (v << 1) ^ P : (v << 1));
   v = (v & 0x80 ? (v << 1) ^ P : (v << 1));
   v = (v & 0x80 ? (v << 1) ^ P : (v << 1));
   v = (v & 0x80 ? (v << 1) ^ P : (v << 1));
   v = (v & 0x80 ? (v << 1) ^ P : (v << 1));
   v = (v & 0x80 ? (v << 1) ^ P : (v << 1));
   return v;
}
uint8_t crc8(const void* data, size_t len, uint8_t v)
{
   const uint8_t* p = (const uint8_t*)data;
   while (len--)
      v = crc_byte((v ^ *p++),POLYNOMIAL);
   return v;
}
```

Given a 10-byte measurement stored at some address, the CRC can be computed over the first 9 bytes as follows:

```
uint8_t crc = crc8(ptr,9,0xFF);
```

Finally, that value can be compared with the CRC stored in the 10th byte of the measurement. Since this is a fairly costly procedure, it is recommended to precompute a 256-entry lookup table instead of calling crc_byte() repeatedly.

It is also recommended that a memory board store all 10-byte readings, including the CRC byte, exactly as received from the transducer without performing any CRC validation. This allows the CRC to continue to protect the measurement while it is stored in the memory board's flash and it reduces the processing power required for the memory board (and therefore extends battery life). When the memory board's contents are later downloaded, host-side software can validate the CRC and potentially display diagnostic information to the user. Specifically, the memory board should NOT check the CRC, store only valid counts in internal flash, protect the counts with a new checksum value and discard the original CRC - the original CRC should always be used to validate a measurement otherwise undetected errors can be introduced in the memory board flash. However, it is acceptable to strip the 0x00 0x55 header since this is common to all measurements, requiring only 8 bytes to be stored in flash per measurement. Since measurements do not include a time

stamp, corrupt measurements cannot simply be discarded since this would cause a shift in the timing for subsequent measurements.

Appendix B
==========
The pressure and temperature polynomials are stored as double-precision floating-point values. These can be retrieved from the transducer using the PLP and PLT commands. These double-precision numbers are represented as their 64-bit hexadecimal values so that they can be transmitted as exact values rather than decimal approximations. In C, these can be converted simply by casting a pointer to the value to the desired type. Using the first hex value in the PLP command example, we have:

```
#include <stdio.h>
#include <stdint.h>

int
main(int argc, const char* argv[])
{
   uint64_t val_hex  = 0x40E5C144C2ED8A82;
   double val_double = *(double*)&val_hex;
   printf("%f\n",val_double);
}
```

Which prints:

```
44554.148795
```

In some cases it may be desirable to talk to the sensor using a scripting language rather than a C program. When using Python, the struct module can be used to convert from a hex value to a floating-point value:

```
>>> import struct
>>> val_hex = 0x40E5C144C2ED8A82
>>> val_double = struct.unpack('d', struct.pack('L', val_hex))[0]
>>> val_double
44554.14879490902
```

Note that the Python float.fromhex() method should not be used since that method requires a different format for the hexadecimal encoding of the number.