Freescale Semiconductor
Application Note

Document Number: AN4519
Rev 0.1, 08/2012

Data Manipulation and Basic Settings of the MPL3115A2 Command Line Interface Driver Code

by: Miguel Salhuana

1 Introduction

It is important to understand how to program the MPL3115A2 to extract pressure and temperature data. The MPL3115A2 has many different features which include 8 different sample rates, 16 different acquisition time steps (1 second to 9 hours), compensated direct reading of Pressure (20 bit in Pascals) or Altitude (20 bit in meters), compensated direct reading of temperature (12 bit in degrees Celsius) and programmable events. It also contains a 32-sample FIFO for collecting and storing data, which gives it the ability to log data for up to 12 days. The FIFO is the most efficient means of accessing the data since it minimizes I^2C transactions. This application note accompanies the MPL3115A2 Command Line Interface Driver Code and will explain how to change the following:

- Modes of operation: Standby, Active Altitude, and Active Barometer
- Sample Rate (OSR)
- Data Acquisition Rate (ST)
- Data Formats (hex to decimal)
- Streaming Pressure/Altitude and Temperature (PT) data polling versus Streaming PT Data with interrupts

Contents

- Using the FIFO to collect PT data by using Overflow and Watermark modes.
- Altitude/Pressure Threshold Alarm and Window Alarm via interrupts
- Temperature Threshold Alarm and Window Alarm via interrupts

1.1 Key words

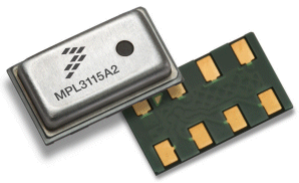
Standby Mode, Active Mode, Altimeter Mode, Barometer Mode, Hexadecimal Numbers, Decimal Numbers, Data Formats, Streaming Data, Polling, Interrupts, FIFO Data, Sensor Toolbox Demo Board, Driver Code, Pressure

1.2 Summary

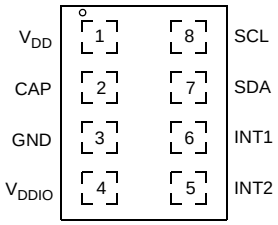
- There are three modes: Standby Mode, Active Altimeter Mode and Active Barometer Mode.
- An example of how to set the data rate (OSR) and the time step (ST) is shown. There are eight OSRs and the time step can be set from 1 second to 9 hours in steps of powers of two.
- Examples of how to setup the sensor to poll data or use an Interrupt Service Routine (ISR) to retrieve data.
- Use of the FIFO in either Overflow or Watermark mode.
- Setup the programmable alarms for Pressure/Altitude with and without specifying a window.
- Setup the programmable alarms for Temperature with and without specifying a window.
- Read and write registers.

2 MPL3115A2 I²C Precision Altimeter

The MPL3115A2 employs a MEMS pressure sensor with an I²C interface to provide accurate Pressure or Altitude data. The sensor's Pressure/Altitude and Temperature outputs are digitized by a high resolution 24-bit ADC. Internal processing removes the compensation tasks from the host MCU system.



LGA Package
5.0 mm by 3.0 mm by 1.1 mm



Top View

Figure 1. Package and pinout diagram

2.1 Key features

- 1.95V to 3.6V Supply voltage
- 1.6V to 3.6V Digital interface supply voltage
- Fully compensated internally
- Direct reading, compensated
  - Pressure: 20-bit measurement (Pascals)
  - Altitude: 20-bit measurement (meters)
  - Temperature: 12-bit measurement (degrees Celsius)
- Programmable events
- Autonomous data acquisition
- Resolution down to 1 foot /30 cm
- 32-sample FIFO
- Ability to log data up to 12 days using the FIFO
- I²C digital output interface (operates up to 400 kHz)

2.2 Two programmable interrupt pins for eight interrupt sources

- Data ready interrupt
- Embedded 32-sample FIFO interrupt
- Pressure/Altitude window alarm interrupt
- Temperature window alarm interrupt
- Pressure/Altitude threshold alarm interrupt
- Temperature threshold alarm interrupt
- Pressure change interrupt
- Temperature change interrupt

3 Modes of Operation

The MPL3115A2 has three modes: Standby, Active Altimeter and Active Barometer. The Standby/Active modes are controlled by bit 0 of the System Control Register (CTRL_REG_1), named SBYB. To switch between the Altimeter and Barometer functionality, flip the ALT bit of CTRL_REG_1.

CTRL_REG_1								
	7	6	5	4	3	2	1	0
R	ALT	RAW	OS2	OS1	OS0	0	OST	SBYB
W						RST		
Reset	0	0	0	0	0	0	0	0

Figure 2. System Control Register 1

3.1 Standby mode

The MPL3115A2 goes into standby mode when a 0 is written to the Standby-Bar, SBYB (bit 0) of CTRL_REG_1. All register writes must be done with the part in Standby mode. The SBYB bit can be written to in Active mode to put the part into Standby mode.

Example 1.

```
/*
** Read contents of CTRL_REG_1
** Clear SBYB mask while holding all other values of CTRL_REG_1.
*/
  CTRL_REG_1_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG1);
  IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, CTRL_REG_1_DATA & STANDBY_SBYB_MASK);
```

3.2 Active Altimeter mode

In order to enter Altimeter Active mode, the MPL3115A2 must be put into standby mode prior to any changes. You must then write a 1 to the ALT bit of CTRL_REG_1 and a 1 to SBYB bit to go into active mode. These writes to the ALT and SBYB can be done in a single I^2C transaction.

Example 2.

```
/*
** Read contents of CTRL_REG_1
** Clear SBYB mask while holding all other values of CTRL_REG_1.
** To put part into Standby mode
*/
  CTRL_REG_1_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG1);
  IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, CTRL_REG_1_DATA & STANDBY_SBYB_MASK);
/*
** Write a 1 to the ALT and SBYB bits to go into Active Altimeter mode
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, (CTRL_REG_1_DATA | ALT _MASK |
ACTIVE_MASK));
```

NOTE

In Altimeter mode, the 20-bit value stored in OUT_P_MSB, OUT_P_CSB and OUT_P_LSB is in meters represented as a signed 16-bit value in OUT_P_MSB and OUT_P_CSB and unsigned fractional value in bits 7-4 of OUT_P_LSB. Bits 3-0 of OUT_P_LSB are unused in this mode.

3.3 Active Barometer mode

In order to enter Active Barometer mode, the MPL3115A2 must be put into Standby mode prior to any changes. You must then write a 0 to the ALT bit of CTRL_REG_1 and a 1 to SBYB bit to go into Active mode.

Example 3.

```
/*
** Read contents of CTRL_REG_1
** Clear SBYB mask while holding all other values of CTRL_REG_1.
** To put part into Standby mode
*/
  CTRL_REG_1_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG1);
  IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, CTRL_REG_1_DATA & STANDBY_SBYB_MASK);

/*
** Write a 0 to the ALT and a 1 SBYB bits to go into Active Barometer mode
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, (CTRL_REG_1_DATA | BAR_MASK | ACTIVE_MASK));
```

NOTE

In Barometer mode, the 20-bit value stored in OUT_P_MSB, OUT_P_CSB and OUT_P_LSB is in Pascals represented as an unsigned 18-bit value in OUT_P_MSB, OUT_P_CSB and bits 7-6 of OUT_P_LSB with the fractional value in bits 5-4 of OUT_P_LSB. Bits 3-0 of OUT_P_LSB are unused in this mode.

4 Setting the Data Rate

4.1 Setting the Output Sample Rate (OSR)

The OSR is programmable via the OS[2:0] in CTRL_REG1 see Figure 3.

CTRL_REG_1								
	7	6	5	4	3	2	1	0
R	ALT	RAW	OS2	OS1	OS0	0	OST	SBYB
W						RST		
Reset	0	0	0	0	0	0	0	0

Figure 3. System Control Register 1

Example 4.

```
/*
** Read contents of CTRL_REG_1
** Clear SBYB mask while holding all other values of CTRL_REG_1.
** To put part into Standby mode
*/
  CTRL_REG_1_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG1);
  IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, CTRL_REG_1_DATA & STANDBY_SBYB_MASK);
/**
 ** The OSR_Value is set to 0 - 7 corresponding with Ratios 1 - 128
*/
if (OSR_Value < 8) {
   OSR_Value <<= 3;
   IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, (CTRL_REG_1_DATA |OSR_Value));
}
```

4.2 Setting the Time Step (ST)

The Auto Acquisition Time Step is programmable via the ST[2:0] control bits in the CTRL_REG_2 illustrated in Figure 4 and Table 1.

CTRL_REG_2								
	7	6	5	4	3	2	1	0
R	0	0	LOAD_OUTPUT	ALARM_SEL	ST[3]	ST[2]	ST[1]	ST[0]
W								
Reset	0	0	0	0	0	0	0	0

Figure 4. System Control Register 2

Table 1. Auto Acquisition Time Step

ST[3]	ST[2]	ST[1]	ST[0]	Time Step (seconds)
0	0	0	0	1
0	0	0	1	2
0	0	1	0	4
0	0	1	1	8
0	1	0	0	16
0	1	0	1	32
0	1	1	0	64
0	1	1	1	128
1	0	0	0	256
1	0	0	1	512
1	0	1	0	1024
1	0	1	1	2048
1	1	0	0	4096
1	1	0	1	8192
1	1	1	0	16384
1	1	1	1	32768

Example 5.

```
/*
** Read contents of CTRL_REG_1
** Clear SBYB mask while holding all other values of CTRL_REG_1.
** To put part into Standby mode
*/
  CTRL_REG_1_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG1);
  IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, CTRL_REG_1_DATA & STANDBY_SBYB_MASK);
/**
 ** The ST_Value is set from 0x1 - 0xF corresponding steps 1 - 32,768
*/
if (ST_Value <= 0xF) {
   CTRL_REG_2_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG2);
   IIC_RegWrite(SlaveAddressIIC, CTRL_REG2, (CTRL_REG_2_DATA|ST_Value));
}
```

5 Data Streaming and Data Conversions

The MPL3115A2 can provide 20-bit pressure data in meters or Pascals. This section is an overview of how to manipulate that data to continuously stream out this 20-bit data. The pressure/temperature data configuration register, PT_DATA_CFG (0x13), has control bits used to enable the internal event flag upon detection of new data, which would occur at the selected OSR. At least one of the data ready enable bits (DREM, PDEFE or TDEFE shown in Figure 4) must be set to active so that an event flag will occur when an updated sample is ready.

PT_DATA_CFG								
	7	6	5	4	3	2	1	0
R						DREM	PDEFE	TDEFE
W								
Reset	0	0	0	0	0	0	0	0

Figure 5. Pressure Data Configuration

The following line of code illustrates how to set the event flag to be generated when a new pressure/altitude reading is available to be read:

```
IIC_RegWrite(SlaveAddressIIC, PT_DATA_CFG_REG, PDEFE_MASK );
```

Once configured, the event flag can be monitored by reading the STATUS register at address 0x00/0x06. This can be done by using either a polling or interrupt technique, which is discussed later in Section 6, “Polling Data Versus Interrupts” of this document. Regardless of the technique used, the STATUS register needs to be read and the appropriate flag monitored.

DR_STATUS								
	7	6	5	4	3	2	1	0
R	PTOW	POW	TOW	0	PTDR	PDR	TDR	0
W								
Reset	0	0	0	0	0	0	0	0

Figure 6. Pressure Data Status

The PDR flag is set whenever there is new pressure data available. The following code example monitors this flag and upon detection of new data, reads the 20-bit Pressure data into three separate 8-bit variables for further processing.

Example 6.

```
RegisterFlag.Byte = IIC_RegRead(SlaveAddressIIC, DR_STATUS_00_REG);

if (RegisterFlag.PDR_BIT == 1)
{
   /*
    ** Read 20 bit Pressure Data Results
    ** and store in 8 bit registers
    */
   pressure_value_raw.Byte.tp_msb = IIC_RegRead(SlaveAddressIIC, OUT_P_MSB_REG);
   pressure_value_raw.Byte.tp_csb = IIC_RegRead(SlaveAddressIIC, OUT_P_CSB_REG);
   pressure_value_raw.Byte.tp_lsb = IIC_RegRead(SlaveAddressIIC, OUT_P_LSB_REG);
}
```

NOTE

In RAW mode the pressure value is stored in all 24 bits that comprise OUT_P_MSB, OUT_P_CSB and OUT_P_LSB; and for the temperature value all 16 bits of OUT_T_MSB and OUT_T_LSB are utilized.

5.1 Converting a 20-bit Altitude reading to a signed decimal number

The 20-bit measurement in meters is comprised of a signed integer component and a fractional component. The signed 16-bit integer component located in OUT_P_MSB and OUT_P_CSB. The fraction component is located in bits 7-4 of OUT_P_LSB. Bits 3-0 of OUT_P_LSB are not used.

The sign of the result is easy to determine by simply checking if the high byte of the value is greater than 0x7F. If so, then the value is a negative number and needs to be transformed by performing a 2's complement conversion. This involves executing a 1's complement (i.e., switch all 1's to 0's and all 0's to 1's) and followed by adding 1 to the result. The following code outputs the data in this format:

Example 7.

```
void SCI_s16dec_Out (tword data)
{
  byte a, b, c, d;
  word r;
  /*
  **  Determine sign and output
  */
  if (data.Byte.hi > 0x7F)
  {
    SCI_CharOut ('-');
    data.Word = ~data.Word + 1;
  }
  else
  {
    SCI_CharOut ('+');
   }
   /*
   **  Calculate
   */
   a = (byte)((data.Word) / 1000);
  r = (data.Word) % 1000;
   b = (byte)(r / 100);
   r %= 100;
   c = (byte)(r / 10);
   d = (byte)(r % 10);
   /*
   **  Format
   */
   if (a == 0)
   {
     a = 0xF0;
     if (b == 0)
     {
       b = 0xF0;
       if (c == 0)
       {
         c = 0xF0;
       }
     }
   }
   /*
   **  Output result
   ** The SCI_NibbOut function outputs a SCI single hex nibble character
   */
   SCI_NibbOut (a);
   SCI_NibbOut (b);
   SCI_NibbOut (c);
   SCI_NibbOut (d);
}
```

To convert the fractional component in OUT_P_LSB, we use the upper byte of the value. Since the resolution is in 0.0625 meters, we can compute the fractional value by using the following table.

Table 2. Fraction Computation

Bit 7	Bit 6	Bit 5	Bit 4	Assigned Value
1	0	0	0	0.5
0	1	0	0	0.25
0	0	1	0	0.125
0	0	0	1	0.0625

We can then add these values together for each bit that is a 1 and we get the fractional component of the altitude value in fractions of a meter. For example, the value of 0.75 meters will be represented by the following 4-bit value: 1100. So using the table above, you would add 0.5 to 0.25 to get 0.75 meters.

Example 8.

```
void SCI_s4fracun_Out (tword data)
{
  BIT_FIELD value;
  word result;
  byte a, b, c, d;
  word r;

  SCI_CharOut ('.');
  /*
  **  Determine mantissa value
  */
  result = 0;
  value.Byte = data.Byte.hi;
  if (value.Bit._7 == 1)
    result += FRAC_2d1;
  if (value.Bit._6 == 1)
    result += FRAC_2d2;
  if (value.Bit._5 == 1)
    result += FRAC_2d3;
  if (value.Bit._4 == 1)
    result += FRAC_2d4;
  //

  /*
  **  Convert mantissa value to 4 decimal places
  */
  r = result % 1000;
  a = (byte)(result / 1000);
  b = (byte)(r / 100);
  r %= 100;
  c = (byte)(r / 10);
  d = (byte)(r % 10);
  /*
  **  Output mantissa
  */
  SCI_NibbOut (a);
  SCI_NibbOut (b);
  SCI_NibbOut (c);
  SCI_NibbOut (d);
}
```

5.2 Converting a 20-bit barometric pressure reading to a decimal value

The 20-bit measurement in Pascals is comprised of an unsigned integer component and a fractional component. The unsigned 18-bit integer component is located in OUT_P_MSB, OUT_P_CSB and bits 7-6 of OUT_P_LSB. The fractional component is located in bits 5-4 of OUT_P_LSB. Bits 3-0 of OUT_P_LSB are not used.

In order to compute the decimal value of the pressure the 20-bit value contained in OUT_P_MSB, OUT_P_CSB and OUT_P_LSB is shifted to the right six places. We can then compute the 18-bit hexadecimal value to an unsigned decimal. The following code achieves this:

Example 9.

```
void SCI_s18decun_Out (tbar_18 data)
{
  byte a, b, c, d, e, f, g, h;
  dword r;
  /*
  **  Calculate
  */

  a = (byte)((data.LWord>>6) / 10000000);     //shift by 6

  r = (data.LWord>>6) % 10000000;

  b = (byte)(r / 1000000);

  r %= 1000000;

  c = (byte)(r / 100000);

  r %= 100000;

  d = (byte)(r / 10000);

  r %= 10000;

  e = (byte)(r / 1000);

  r %= 1000;

  f = (byte)(r / 100);

  r %= 100;

  g = (byte)(r / 10);

  h = (byte)(r % 10);


  /*
```

```
  **  Format
  */
  if (a == 0)
  {
    a = 0xF0;
    if (b == 0)
    {
      b = 0xF0;
      if (c == 0)
      {
        c = 0xF0;
        if (d == 0) {
          d = 0xF0;
          if (e==0){
            e = 0xF0;
            if (f == 0){
              f = 0xF0;
              if ( g == 0 ) {
                g = 0xF0;
              }
            }
          }
        }
      }
    }
  }
  /*
  **  Output result
  */
  SCI_NibbOut (d);
  SCI_NibbOut (e);
  SCI_NibbOut (f);
  SCI_NibbOut (g);
  SCI_NibbOut (h);
}
```

To convert the fractional component contained in bits 5-4 of OUT_P_LSB we use the upper byte of the value and shift it to the left by two places. The resolution is in 0.25 Pascals we can compute the fractional value by using the following table:

Table 3. Fraction Computation

Bit 5	Bit 4	Assigned Value
0	1	0.5
1	0	0.25

We can then add these values together for each bit that is a 1 and we get the fractional component of the altitude value in meters. So for a value of 0.75 bit 5 and bit 4 will both be set to 1 which will give you 0.5 + 0.25 = 0.75.

Example 10.

```
void SCI_s2fracun_Out (tword data)
{
  BIT_FIELD value;
  word result;
  byte a, b, c, d;
  word r;

  SCI_CharOut ('.');
  /*
  **  Determine mantissa value
  */
  result = 0;
  value.Byte = data.Byte.hi;
  if (value.Bit._5 == 1)
    result += FRAC_2d1;
  if (value.Bit._4 == 1)
    result += FRAC_2d2;

  /*
  **  Convert mantissa value to 2 decimal places
  */
  r = result % 1000;
  a = (byte)(result / 1000);
  b = (byte)(r / 100);
  r %= 100;
  c = (byte)(r / 10);
  d = (byte)(r % 10);
  /*
  **  Output mantissa
  */
  SCI_NibbOut (a);
  SCI_NibbOut (b);
}
```

5.3 Converting a 16-bit temperature reading to a signed decimal number

The 12-bit temperature measurement in degrees Celsius is comprised of a signed integer component and a fractional component. The signed 8-bit integer component is located in OUT_T_MSB. The fractional component is located in bits 7-4 of OUT_T_LSB. Bits 3-0 of OUT_T_LSB are not used.

The sign of the result is easy to determine by simply checking if the high byte of the value is greater than 0x7F. If so, then the value is a negative number and needs to be transformed by performing a 2's complement conversion. This involves executing a 1's complement (i.e., switch all 1's to 0's and all 0's to 1's) and followed by adding 1 to the result. The following code outputs the data in this format:

Example 11.

```
void SCI_s8dec_Out (tword data)
{
  byte a, b, c;
  word r;
  /*
  **  Determine sign and output
  */
  if (data.Byte.hi > 0x7F)
  {
    SCI_CharOut ('-');
    data.Word = ~data.Word + 1;
  }
  else
  {
    SCI_CharOut ('+');
  }
  /*
  **  Calculate
  */
  a = (byte)((data.Word >>8) / 100); //Shift the data over since it is MSB only
  r = (data.Word >>8) % 100;
  b = (byte)(r / 10);
  c = (byte)(r % 10);
  /*
  **  Format
  */
  if (a == 0)
  {
    a = 0xF0;
    if (b == 0)
    {
      b = 0xF0;
    }
  }
  /*
  **  Output result
  */
  SCI_NibbOut (a);
  SCI_NibbOut (b);
  SCI_NibbOut (c);
}
```

The conversion of the fractional component is done using the same methodology as in converting fractions of a meter illustrated in Section 5.1, "Converting a 20-bit Altitude reading to a signed decimal number".

6 Polling Data Versus Interrupts

The data can be polled continuously or a hardware interrupt can be configured to signal when the data ready flag is set. The MCU can then use an interrupt service routine to retrieve the data. Depending on the circumstances, one might be more desirable than the other although polling typically is less efficient.

6.1 Polling data

Polling requires less configuration of the device and is very simple to implement. However, the MCU must poll the sensor at a rate that is faster than the output data rate. Otherwise, if the polling is too slow, data samples will be missed. The MCU can detect the missed or corrupted data condition by checking the overwrite flags in the STATUS register (i.e., PTOW, POW, TOW, PTDR, PDR, and TDR). The code examples provided so far in this document have primarily described the polling technique. As a summary, here is a more complete example of the basic code, specific to the operation of the MPL3115A2, required to continuously poll 20-bit PT data using the Altimeter Active mode:

Example 12.

```
/*
** Read contents of CTRL_REG_1
** Clear SBYB mask while holding all other values of CTRL_REG_1.
** To put part into Standby mode
*/
  CTRL_REG_1_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG1);
  IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, CTRL_REG_1_DATA & STANDBY_SBYB_MASK);
/*
** Write a 1 to the ALT and SBYB bits to go into Active Altimeter mode
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, (CTRL_REG_1_DATA | ALT _MASK |
ACTIVE_MASK));

/*
** Using a basic control loop, continously poll the sensor
*/

for (;;) {
   /**
    ** Poll the
   */
   RegisterFlag.Byte = IIC_RegRead(SlaveAddressIIC, DR_STATUS_00_REG);
   if (RegisterFlag.PDR_BIT == 1)
      if (RegisterFlag.TDR_BIT == 1) {
         SCISendString(" Hex ");
         temp =   IIC_RegRead(SlaveAddressIIC, OUT_P_MSB_REG);
         a_mcsb_value.Byte.hi = temp;
         SCI_ByteOut(temp);
         temp = IIC_RegRead(SlaveAddressIIC, OUT_P_CSB_REG);
         a_mcsb_value.Byte.lo = temp;
         SCI_ByteOut(temp);
         temp = IIC_RegRead(SlaveAddressIIC, OUT_P_LSB_REG);
         a_dec_value.Byte.lo = 0x0;
```

```
        a_dec_value.Byte.hi = temp;
        SCI_ByteOut(temp);
        SCISendString(" Dec ");
        SCI_s16dec_Out(a_mcsb_value);
        SCI_s4fracun_Out(a_dec_value);
        SCISendString(" Hex ");
        temp =   IIC_RegRead(SlaveAddressIIC, OUT_T_MSB_REG);
        SCI_ByteOut(temp);
        t_msb_value.Byte.hi = temp;
        t_msb_value.Byte.lo = 0x0;
        temp = IIC_RegRead(SlaveAddressIIC, OUT_T_LSB_REG);
        t_lsb_value.Byte.hi = temp;
        t_lsb_value.Byte.lo = 0x0;
        SCI_ByteOut(temp);
        SCISendString(" Dec ");
        SCI_s8dec_Out(t_msb_value);
        SCI_s4fracun_Out(t_lsb_value);
    }
}
}
```

6.2 Interrupt routine to access data

Streaming data via hardware interrupts is more efficient than polling as the MCU only interfaces with the MPL3115A2 when it has new data. The output registers are read only when new data is available. If the data is not read every time, the acquisition of new data will be indicated by the overwrite register flags. The following are the register settings to configure the MPL3115A2 to generate an interrupt upon each new Pressure and Temperature (PT) sample in Altimeter mode. The MCU's Interrupt Service Routine (ISR) shown below responds by reading the 20-bit Pressure and 12-bit Temperature data and setting a software flag indicating the arrival of new data. It is considered to be good practice to keep ISRs as fast as possible, so the actual processing of this data is not done here.

Please be aware that when enabling/disabling interrupts you should clear all existing interrupts.

Example 13.

```
/*
** Read contents of CTRL_REG_1
** Clear SBYB mask while holding all other values of CTRL_REG_1.
** To put part into Standby mode
*/
  CTRL_REG_1_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG1);
  IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, CTRL_REG_1_DATA & STANDBY_SBYB_MASK);

/*
** Clear all interrupts by reading the output registers.
** Enable the Data Ready Interrupt and route to INT 1
** Activate sensor in Altimeter mode.
*/
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_CSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, F_STATUS);

IIC_RegWrite(SlaveAddressIIC, CTRL_REG4, INT_EN_DRDY_MASK);
IIC_RegWrite(SlaveAddressIIC, CTRL_REG5, INT_CFG_DRDY_MASK);

/*
 ** Configure the INT pins for Open Drain and Active Low.
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG3, (PP_OD1_MASK | PP_OD2_MASK));
/*
** Write a 1 to the ALT and SBYB bits to go into Active Altimeter mode
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, (CTRL_REG_1_DATA | ALT _MASK |
ACTIVE_MASK));



interrupt void isr_MPL3115A2 (void)
{
  /*
  **  Clear the interrupt flag
  */
  CLEAR_MPL3115A2_INTERRUPT;
  /** setting CHECK_INT flag to true
   ** allows us to handle the data manipulation
   ** in the main loop
  */
  CHECK_INT = TRUE;
}
```

7 Pressure/Altitude Alarms Interrupts

The MPL3115A2 provides two Pressure/Altitude alarm interrupts, a threshold alarm and a window alarm.

For the Threshold alarm the interrupt flag is set on reaching the value stored in the Pressure/Altitude target register (P_TGT). Additionally, a window value (P_WND) provides the ability to signal when the target is nearing the lower and upper threshold values set as offsets from the Pressure/Altitude target value. When in barometer mode, these values represent pressures rather than altitudes.

Examples:

- Set Altitude alert to 3000m and window value to 100m, interrupt is asserted passing 2900m, 3000m, and 3100m.
- Set Pressure alert to 100.0 kPa and window value to 5 kPa, interrupt is asserted passing 95 kPa, 100 kPa and 105 kPa.

The Window alarm interrupt flag is set when the temperature value is within the window defined by the following equation:

$$Window = P\_TGT_{MSB,LSB} \pm P\_WND_{MSB,LSB} \quad \textit{Eqn. 1}$$

7.1 Altitude alarms

In Altitude mode the target value in P_TGT_(MSB,LSB) is a signed 16-bit 2's complement value in meters. The window value in P_WND(MSB,LSB) is an unsigned 16-bit value in meters.

7.1.1 Setting threshold alarm for target Altitude

To set the altitude alarm so that the interrupt goes off only at the target altitude, we must set the P_TGT_MSB and P_TGT_LSB registers to our target (in meters) and set the window register P_WND_MSB and P_WND_LSB to zero. We then enable the Pressure threshold interrupt by writing a 1 to bit 3 (INT_EN_PTH) of CTRL_REG4 (Interrupt Enable register) and routing the interrupt to pin INT2 by writing a zero to bit 3 (INT_CFG_PTH) of CTRL_REG5 (Interrupt Configuration register).

CTRL_REG4								
	7	6	5	4	3	2	1	0
R	INT_EN_DRDY	INT_EN_FIFO	INT_EN_PW	INTE_EN_TW	INT_EN_PTH	INT_EN_TTH	INT_EN_PCHG	INT_EN_TCHG
W								
Reset	0	0	0	0	0	0	0	0

Figure 7. Control Register 4

CTRL_REG5 (0x2A)								
	7	6	5	4	3	2	1	0
R	INT_CFG_DRDY	INT_CFG_FIFO	INT_CFG_PW	INTE_CFG_TW	INT_CFG_PTH	INT_CFG_TTH	INT_CFG_PCHG	INT_CFG_TCHG
W								
Reset	0	0	0	0	0	0	0	0

Figure 8. Control Register 5

Example 14.

```
** Clear SBYB mask while holding all other values of CTRL_REG_1.
** To put part into Standby mode
*/
  CTRL_REG_1_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG1);
  IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, CTRL_REG_1_DATA & STANDBY_SBYB_MASK);

/*
** Clear all interrupts by reading the output registers.
** Enable the Data Ready Interrupt and route to INT 1
** Activate sensor in Altimeter mode.
*/
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_CSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, F_STATUS);

/*
** Write Target and Window Values
** value[3] and value[4] are set to 0
*/
IIC_RegWrite(SlaveAddressIIC, P_TGT_MSB, value[1]);
IIC_RegWrite(SlaveAddressIIC, P_TGT_LSB, value[2]);
IIC_RegWrite(SlaveAddressIIC, P_TGT_WND_MSB, value[3]);
IIC_RegWrite(SlaveAddressIIC, P_TGT_WND_LSB, value[4]);

/*
** Enable Interrupt and Map to Interrupt Pin
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG4, INT_EN_PTH_MASK);
IIC_RegWrite(SlaveAddressIIC, CTRL_REG5, INT_CFG_PTH_MASK);
/*
** Write a 1 to the ALT and SBYB bits to go into Active Altimeter mode
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG1,(CTRL_REG_1_DATA | ALT_MASK | ACTIVE MASK));

interrupt void isr_MPL3115A2 (void)
{
```

```
  /*
  **  Clear the interrupt flag
  */
  CLEAR_MPL3115A2_INTERRUPT;
  /** setting CHECK_INT flag to true
   ** allows us to handle the data interrupt
   ** in the main loop
  */
  CHECK_INT = TRUE;
}
```

7.1.2 Setting Threshold Alarm for upper and lower thresholds

To set the Altitude alarm so that the interrupt goes off at the target altitude as well as at the lower and upper thresholds, we must set the P_TGT_MSB and P_TGT_LSB registers to our target in meters and set the window register P_WND_MSB and P_WND_LSB to the desired offset. We then enable the Pressure threshold interrupt by writing a 1 to bit 3 (INT_EN_PTH) of CTRL_REG4 (Interrupt Enable register) and routing the interrupt to pin INT2 by writing a zero to bit 3 (INT_CFG_PTH) of CTRL_REG5 (Interrupt Configuration register).

Example 15.

```
** Clear SBYB mask while holding all other values of CTRL_REG_1.
** To put part into Standby mode
*/
  CTRL_REG_1_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG1);
  IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, CTRL_REG_1_DATA & STANDBY_SBYB_MASK);

/*
** Clear all interrupts by reading the output registers.
** Enable the Data Ready Interrupt and route to INT 1
** Activate sensor in Altimeter mode.
*/
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_CSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, F_STATUS);

/*
** Write Target and Window Values
** value[3] and value[4] are set to the offset
*/
IIC_RegWrite(SlaveAddressIIC, P_TGT_MSB, value[1]);
IIC_RegWrite(SlaveAddressIIC, P_TGT_LSB, value[2]);
IIC_RegWrite(SlaveAddressIIC, P_TGT_WND_MSB, value[3]);
IIC_RegWrite(SlaveAddressIIC, P_TGT_WND_LSB, value[4]);
```

```
/*
** Enable Interrupt and Map to Interrupt Pin
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG4, INT_EN_PTH_MASK);
IIC_RegWrite(SlaveAddressIIC, CTRL_REG5, INT_CFG_PTH_MASK);
/*
** Write a 1 to the ALT and SBYB bits to go into Active Altimeter mode
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, (CTRL_REG_1_DATA | ALT_MASK | ACTIVE MASK));
interrupt void isr_KBI (void)
{
  /*
  **  Clear the interrupt flag
  */
  CLEAR_KBI_INTERRUPT;
  /** setting CHECK_INT flag to true
   ** allows us to handle the data interrupt
   ** in the main loop
  */
  CHECK_INT = TRUE;
}
```

7.1.3 Setting the window alarm

To set the Altitude alarm so that the interrupt goes off only when we are in the window defined by P_TGT± P_WND we must set the P_TGT_MSB and P_TGT_LSB registers to our target in meters and set the window register P_WND_MSB and P_WND_LSB the desired offset. We then enable the Pressure window interrupt by writing a 1 to bit 5 (INT_EN_PW) of CTRL_REG4 (Interrupt Enable register) and routing the interrupt to pin INT2 by writing a zero to bit 5 (INT_CFG_PW) of CTRL_REG5 (Interrupt Configuration register).

Example 16.

```
** Clear SBYB mask while holding all other values of CTRL_REG_1.
** To put part into Standby mode
*/
  CTRL_REG_1_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG1);
  IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, CTRL_REG_1_DATA & STANDBY_SBYB_MASK);

/*
** Clear all interrupts by reading the output registers.
** Enable the Data Ready Interrupt and route to INT 1
** Activate sensor in Altimeter mode.
*/
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_CSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, F_STATUS);
```

```
/*
** Write Target and Window Values
** value[3] and value[4] are set to the offset
*/
IIC_RegWrite(SlaveAddressIIC, P_TGT_MSB, value[1]);
IIC_RegWrite(SlaveAddressIIC, P_TGT_LSB, value[2]);
IIC_RegWrite(SlaveAddressIIC, P_TGT_WND_MSB, value[3]);
IIC_RegWrite(SlaveAddressIIC, P_TGT_WND_LSB, value[4]);

/*
** Enable Interrupt and Map to Interrupt Pin
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG4, INT_EN_PW_MASK);
IIC_RegWrite(SlaveAddressIIC, CTRL_REG5, INT_CFG_PW_MASK);
/*
** Write a 1 to the ALT and SBYB bits to go into Active Altimeter mode
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG1,(CTRL_REG_1_DATA | ALT_MASK | ACTIVE_MASK));

interrupt void (void)
{
  /*
  **  Clear the interrupt flag
  */
  CLEAR_MPL3115A2_INTERRUPT;
  /** setting CHECK_INT flag to true
   ** allows us to handle the data interrupt
   ** in the main loop
  */
  CHECK_INT = TRUE;
}
```

7.2 Pressure alarms

The Pressure alarms work exactly the same as the Altitude alarms in Section 7.1, “Altitude alarms” with the difference being that the values of P_TGT_(MSB,LSB) and P_WND_(MSB,LSB) are unsigned 16-bit values in 2 Pascal units.

8 Temperature Alarm Interrupts

The MPL3115A2 provides two interrupt alarms for Temperature, a threshold alarm and a window alarm.

For the Threshold alarm, the interrupt flag is set on reaching the value stored in the Temperature target register (T_TGT). Additionally, a window value (T_WND) provides the ability to signal when the target is nearing the lower threshold, the target threshold or the upper threshold value set as offsets from the Temperature target value.

The T_TGT register holds the Temperature target value, in °C, as a signed 8-bit value in 2's complement. The T_WND register holds the offset value as an unsigned 8-bit value.

Examples:

- Set Temperature alert to 30°C and window value to 10°C, interrupt is asserted passing 20°C, 30°C, and 40°C.
- The Window alarm interrupt flag is set when the temperature value is within the window defined by the following equation:

$$Window = T\_TGT \pm T\_WND \quad \textit{Eqn. 2}$$

8.1 Setting threshold alarm for target temperature

To set the temperature alarm so that the interrupt goes off only at the target temperature we must set the T_TGT register to our target in °C and set the window register T_WND to zero. We then enable the Temperature threshold interrupt by writing a 1 to bit 2 (INT_EN_TTH) of CTRL_REG4 (Interrupt Enable register) and routing the interrupt to pin INT2 by writing a zero to bit 2 (INT_CFG_TTH) of CTRL_REG5 (Interrupt Configuration register).

Example 17.

```
** Clear SBYB mask while holding all other values of CTRL_REG_1.
** To put part into Standby mode
*/
  CTRL_REG_1_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG1);
  IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, CTRL_REG_1_DATA & STANDBY_SBYB_MASK);

/*
** Clear all interrupts by reading the output registers.
** Enable the Data Ready Interrupt and route to INT 1
** Activate sensor in Altimeter mode.
*/
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_CSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, F_STATUS);

/*
** Write Target and Window Values
```

```
** value[2] is set to 0
*/
IIC_RegWrite(SlaveAddressIIC, T_TGT, value[1]);
IIC_RegWrite(SlaveAddressIIC, T_WND, value[2]);

/*
** Enable Interrupt and Map to Interrupt Pin
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG4, INT_EN_TTH_MASK);
IIC_RegWrite(SlaveAddressIIC, CTRL_REG5, INT_CFG_TTH_MASK);


/*
** Write a 1 to the SBYB bits to go into Active mode
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, (CTRL_REG_1_DATA | ACTIVE_MASK));


interrupt void isr_MPL3115A2 (void)
{
  /*
  **  Clear the interrupt flag
  */
  CLEAR_KBI_INTERRUPT;
  /** setting CHECK_INT flag to true
   ** allows us to handle the data interrupt
   ** in the main loop
  */
  CHECK_INT = TRUE;
}
```

8.2 Setting threshold alarm for upper and lower thresholds

To set the Temperature alarm so that the interrupt goes off at the target temperature as well as at the lower and upper thresholds we must set the T_TGT register to our target in °C and set the window register T_WND to the desired offset. We then enable the Temperature Threshold interrupt by writing a 1 to bit 2 (INT_EN_TTH) of CTRL_REG4 (Interrupt Enable register) and routing the interrupt to pin INT2 by writing a zero to bit 2 (INT_CFG_TTH) of CTRL_REG5 (Interrupt Configuration register).

Example 18.

```
** Clear SBYB mask while holding all other values of CTRL_REG_1.
** To put part into Standby mode
*/
  CTRL_REG_1_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG1);
  IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, CTRL_REG_1_DATA & STANDBY_SBYB_MASK);

/*
** Clear all interrupts by reading the output registers.
** Enable the Data Ready Interrupt and route to INT 1
** Activate sensor in Altimeter mode.
```

```
*/
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_CSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, F_STATUS);

/*
** Write Target and Window Values
** value[2] are set to the offset
*/
IIC_RegWrite(SlaveAddressIIC, T_TGT, value[1]);
IIC_RegWrite(SlaveAddressIIC, T_WND, value[2]);

/*
** Enable Interrupt and Map to Interrupt Pin
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG4, INT_EN_TTH_MASK);
IIC_RegWrite(SlaveAddressIIC, CTRL_REG5, INT_CFG_TTH_MASK);

/*
** Write a 1 to the SBYB bits to go into Active mode
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, (CTRL_REG_1_DATA | ACTIVE_MASK));

interrupt void isr_MPL3115A2 (void)
{
  /*
  **  Clear the interrupt flag
  */
  CLEAR_KBI_INTERRUPT;
  /** setting CHECK_INT flag to true
   ** allows us to handle the data interrupt
   ** in the main loop
  */
  CHECK_INT = TRUE;
}
```

8.3 Setting the window alarm

To set the Temperature alarm so that the interrupt goes off only when we are in the window defined by T_TGT ± T_WND we must set the T_TGT register to our target in °C and set the window register T_WND to the desired offset. We then enable the Temperature Window interrupt by writing a 1 to bit 4 (INT_EN_TW) of CTRL_REG4 (Interrupt Enable register) and routing the interrupt to pin INT2 by writing a zero to bit 4 (INT_CFG_TW) of CTRL_REG5 (Interrupt Configuration register).

Example 19.

```
** Clear SBYB mask while holding all other values of CTRL_REG_1.
** To put part into Standby mode
*/
  CTRL_REG_1_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG1);
  IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, CTRL_REG_1_DATA & STANDBY_SBYB_MASK);

/*
** Clear all interrupts by reading the output registers.
** Enable the Data Ready Interrupt and route to INT 1
** Activate sensor in Altimeter mode.
*/
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_CSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, F_STATUS);

/*
** Write Target and Window Values
** value[1] and value[2] are set to the offset
*/
IIC_RegWrite(SlaveAddressIIC, T_TGT, value[1]);
IIC_RegWrite(SlaveAddressIIC, T_WND, value[2]);

/*
** Enable Interrupt and Map to Interrupt Pin
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG4, INT_EN_TW_MASK);
IIC_RegWrite(SlaveAddressIIC, CTRL_REG5, INT_CFG_TW_MASK);

/*
** Write a 1 to the SBYB bits to go into Active mode
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, (CTRL_REG_1_DATA | ACTIVE_MASK));

interrupt void isr_MPL3115A2 (void)
{
  /*
  **  Clear the interrupt flag
  */
  CLEAR_KBI_INTERRUPT;
  /** setting CHECK_INT flag to true
   ** allows us to handle the data interrupt
   ** in the main loop
  */
  CHECK_INT = TRUE;
}
```

9 Using the 32-sample FIFO

9.1 Introduction

The MPL3115A2 has a built-in 32-sample first in, first out (FIFO) buffer capable of storing the 20-bit pressure data and the 12-bit Temperature data. The FIFO is very beneficial for saving overall system power by putting the processor into sleep mode until it needs to process data from the sensor. The idea is to configure the MPL3115A2 to monitor a desired interrupt, putting the processor into a low-power mode until it needs to respond to the sensor. This minimizes the system's overall power consumption, increasing the life of the battery. The embedded FIFO is a proven benefit as it limits how often the processor needs to read the data. The FIFO allows the processor to sleep longer while samples are being collected inside the sensor. Higher sample-rate data can be captured in the FIFO and accessed at a reasonable update time without increasing computational throughput by accessing every sample individually.

9.2 Summary

- The embedded FIFO is highly beneficial for system power savings and minimizing traffic across the I^2C bus.
- The FIFO allows for remarkable power savings of the system by allowing the host processor/MCU to go into a sleep mode while the pressure sensor independently stores the data, up to 32 samples.
- The FIFO can be configured to be in a circular buffer mode or a fill buffer mode.
- The FIFO allows you to log data for up to 12 days using the highest ST setting of 9 hours.

9.3 Embedded settings of the FIFO

The following sections discuss the different registers involved in configuring the FIFO.

9.3.1 F_SETUP register (0x0F)

The FIFO mode and the watermark is specified in the F_SETUP register. The FIFO can be configured in either a circular buffer or in overflow mode. In circular buffer mode a watermark can be set to trigger a flag event. Exceeding the watermark count does not stop the FIFO from accepting new data, the oldest data is overwritten.

F_SETUP								
	7	6	5	4	3	2	1	0
R	F_MODE1	F_MODE0	F_WMRK5	F_WMRK4	F_WMRK3	F_WMRK2	F_WMRK1	F_WMRK0
W								
Reset	0	0	0	0	0	0	0	0

Figure 9. FIFO setup register

9.3.2 Setting up the FIFO interrupt

In order to setup the FIFO interrupt event the INT_EN_FIFO bit of CTRL_REG4 must be set and the INT_CFG_FIFO bit of CTRL_REG5 must be set to route the interrupt to INT1 or INT2.

9.3.3 F_STATUS register (0x0D)

The FIFO status register indicates the current status information of the FIFO subsytem.

F_STATUS								
	7	6	5	4	3	2	1	0
R	F_OVF	F_WMRK_FLAG	F_CNT5	F_CNT4	F_CNT3	F_CNT_2	F_CNT_1	F_CNT_0
W								
Reset	0	0	0	0	0	0	0	0

Figure 10. FIFO status register

Table 4. FIFO Flag Event Descriptions

F_OVF	F_WMRK_FLAG	Result
0	—	No FIFO overflow event detected
1	—	FIFO overflow event detected
—	0	No FIFO watermark event detected
—	1	FIFO watermark event detected

The F_OVF and F_WMRK_FLAG remain asserted while the event source is still active, but the user can clear the FIFO interrupt bit flag in the INT_SOURCE register be reading the F_STATUS register. Therefore the F_OVF bit flag will remain asserted while the FIFO has overflowed and the F_WMRK_FLAG bit flag will remain asserted while the F_CNT value is greater than the F_WMRK value.

Table 5. FIFO Sample Count Description

Name	Description
F_CNT[5:0]	FIFO sample counter. Indicates the number of samples currently stored in the FIFO. 000000 indicates that the FIFO is empty 000001 to 100000 indicates 1 to 32 samples stored in the FIFO

9.3.4 F_DATA register (0xE)

F_DATA is a read-only register which provides access to FIFO data. FIFO holds a maximum of 32 P/T samples. Each sample consists of 5 bytes; 3 bytes of pressure data and 2 bytes of temperature data. A maximum of 5 x 32 = 160 data bytes of samples can be stored. When F_MODE bit in FIFO SETUP (F_SETUP) register is set to logic “1”, the F_DATA pointer shares the same address location as OUT_P_MSB (0x01); therefore all accesses of the FIFO buffer data use the I^2C sub register address of 0x01. Reads from the other data registers (0x02, 0x03, 0x04, 0x05) will return a value of 0x00. The FIFO will not suspend data accumulation during reads to F_DATA register. All 160 bytes of data can be read out using a multibyte I^2C read subroutine.

F_DATA 8-bit Data Access Register								
	7	6	5	4	3	2	1	0
R	F_DATA							
W								
Reset	0	0	0	0	0	0	0	0

Figure 11. FIFO Data Register

Table 6. Read accesses through F_DATA

1st Read	OUT_P_MSB (oldest)
2nd Read	OUT_P_CSB (oldest)
3rd Read	OUT_P_LSB (oldest)
4th Read	OUT_T_MSB (oldest)
5th Read	OUT_T_LSB (oldest)
. . .	. . .
	OUT_T_LSB (oldest)
	0x00
	0x00

9.3.5 Setting the interrupts for FIFO

In order to enable the FIFO interrupt you must write a 1 to bit 6 (INT_EN_FIFO) of the interrupt enable register CTRL_REG4 (0x29). The interrupt configuration register CTRL_REG5 controls which interrupt the flag goes to writing a 1 to bit 6 (INT_CFG_FIFO) of this register will send the flag to INT1 pin and writing a 0 will send it to INT2 pin.

9.4 Setting up the FIFO in Overflow mode

To setup the FIFO in overflow mode we will configure the F_MODE[1:0] bits of the F_SETUP(0x0F) register to 10, setup the interrupt flag to go to INT2 and look for the F_OVF bit in the F_STATUS register.

Example 20.

```
/*
** Read contents of CNTL_REG_1
** Clear SBYB mask while holding all other values of CTRL_REG_1.
** To put part into Standby mode
*/
CTRL_REG_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG1);
IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, CTRL_REG_1_DATA & STANDBY_SBYB_MASK);

/*
** Clear all interrupts by reading the output registers.
** Enable the Data Ready Interrupt and route to INT 1
** Activate sensor in Altimeter mode.
*/
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_CSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, F_STATUS);

/*
**  Clear FIFO MODE by writing a 00 to F_MODE[1:0]
*/
IIC_RegWrite(SlaveAddressIIC, F_SETUP_REG, F_CLEAR_MASK);

/*
 ** Setup the FIFO to Overflow Mode
 ** Route to INT2
 ** Open Drain Active Low Interrupts
*/

IIC_RegWrite(SlaveAddressIIC, CTRL_REG3, (PP_OD1_MASK | PP_OD2_MASK));

IIC_RegWrite(SlaveAddressIIC, F_SETUP_REG, F_MODE10_MASK);
IIC_RegWrite(SlaveAddressIIC, CTRL_REG4, INT_EN_FIFO_MASK);
IIC_RegWrite(SlaveAddressIIC, CTRL_REG5, INT_CFG_CLEAR);

/*
```

```
 ** Put part into Active mode
*/

IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, (IIC_RegRead(SlaveAddressIIC, CTRL_REG1) |
ACTIVE_MASK));


/*******************************************************/
/* MPL3115A2 Interrupt Service Routine for the FIFO    /
/*******************************************************/
interrupt void isr_MPL3115A2 (void)
{
  /*
   ** Clear the MCUs interrupt flag
  */
  CLEAR_MPL3155A2_INTERRUPT;
  /*
   ** Go read interrupt source register
  */
  RegisterFlag.Byte = IIC_RegRead(SlaveAddressIIC, INT_SOURCE_REG);
  if(RegisterFlag.SRC_FIFO_BIT == 1)
  {
          RegisterFlag.Byte = IIC_RegRead(SlaveAddressIIC, F_STATUS_REG);
          /*
          **  Go read FIFO with a single multi-byte IIC access
          */
          value[4] = (RegisterFlag.Byte & F_CNT_MASK) * 5;
          IIC_RegReadN(SlaveAddressIIC, OUT_P_MSB_REG, value[4],
&fifo_data[0].Sample.BT.b_msb);
  }


}
```

The interrupt routine dumps the FIFO into an array for further processing. This method is not ideal and is used only to illustrate the multibyte read. The ideal method is to set a flag in the interrupt service routine and then use a task or function to handle the flag.

9.5 Setting the FIFO with a Watermark

To setup the FIFO in watermark mode we will configure the F_MODE[1:0] bits of the F_SETUP(0x0F) register to 01, setup the interrupt flag to go to INT2, write a watermark value to F_WMRK[5:0] of the F_SETUP register and look for the F_WMRK_FLAG bit in the F_STATUS register.

Example 21.

```
/*
** Read contents of CTRL_REG_1
** Clear SBYB mask while holding all other values of CTRL_REG_1.
** To put part into Standby mode
*/
CTRL_REG_1_DATA = IIC_RegRead(SlaveAddressIIC, CTRL_REG1);
IIC_Reg_Write(SlaveAddressIIC, CTRL_REG1, CTRL_REG_1_DATA & STANDBY_SBYB_MASK);
/*
** Clear all interrupts by reading the output registers.
** Enable the Data Ready Interrupt and route to INT 1
** Activate sensor in Altimeter mode.
*/
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_CSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_P_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_MSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, OUT_T_LSB_REG);
temp = IIC_RegRead(SlaveAddressIIC, F_STATUS);

/*
**  Clear FIFO MODE by writing a 00 to F_MODE[1:0]
*/
IIC_RegWrite(SlaveAddressIIC, F_SETUP_REG, F_CLEAR_MASK);

/*
 ** Setup the FIFO to Watermark Mode
 ** Write a watermark value
 ** Route to INT2
 ** Open Drain Active Low Interrupts
*/
IIC_RegWrite(SlaveAddressIIC, CTRL_REG3, (PP_OD1_MASK | PP_OD2_MASK));

value[1] =0x05;

IIC_RegWrite(SlaveAddressIIC, F_SETUP_REG, (F_MODE01_MASK | value[1]));
IIC_RegWrite(SlaveAddressIIC, CTRL_REG4, INT_EN_FIFO_MASK);
IIC_RegWrite(SlaveAddressIIC, CTRL_REG5, INT_CFG_CLEAR);

/*
 ** Put part into Active mode
*/

IIC_RegWrite(SlaveAddressIIC, CTRL_REG1, (IIC_RegRead(SlaveAddressIIC, CTRL_REG1) |
ACTIVE_MASK));
```

We use the same interrupt routine as the Overflow mode to dump the FIFO.

10 Command Line Interface User Guide

The command line interface driver code provides the customer with a starting point to develop their own application specific code. The code was written on the CodeWarrior development suite. Download the code via the BDM to the DEMOSTB MPL3115A2. An image of the demo board is shown:

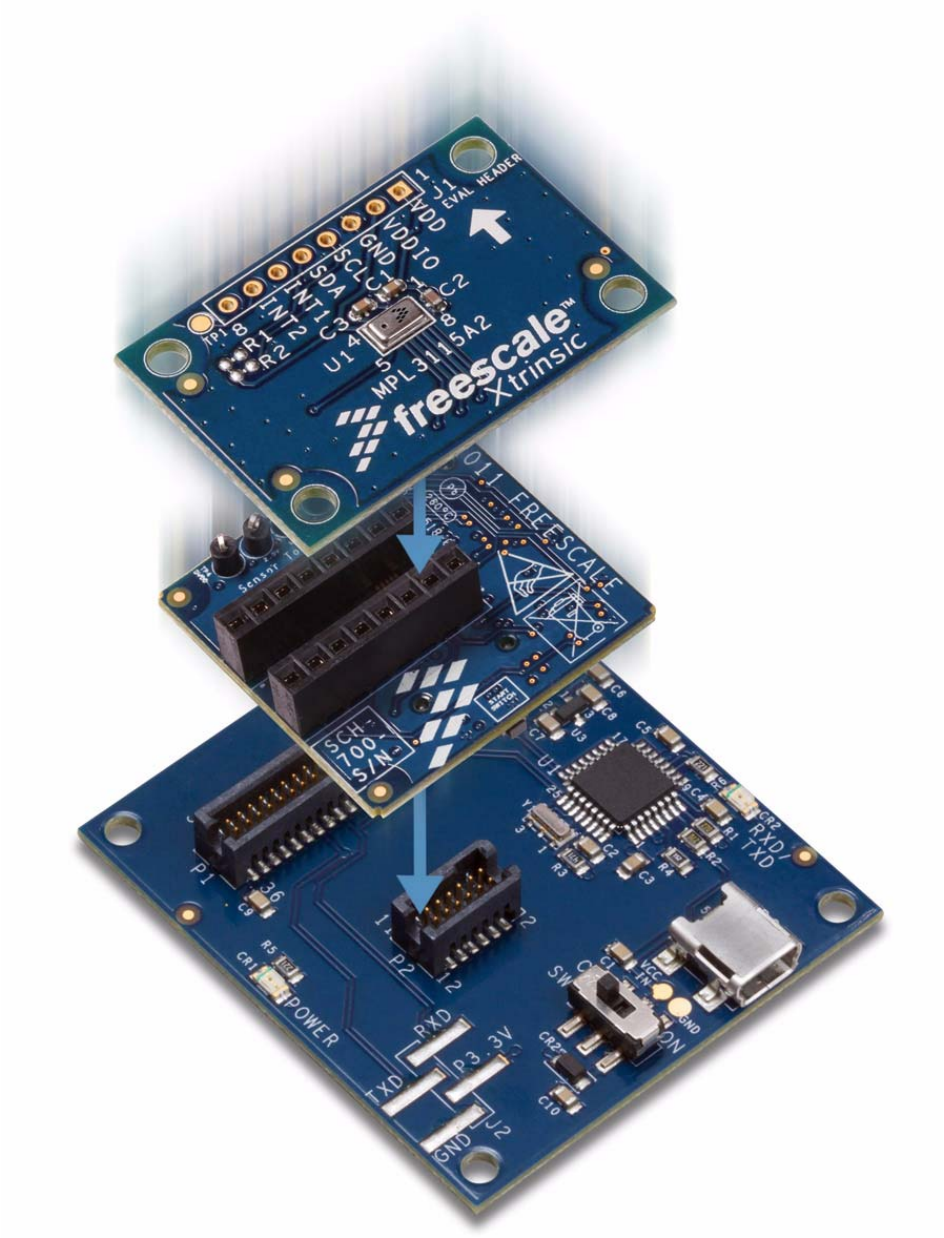


Figure 12. MPL3115A2 daughter board, LFSTBEBMPL3115A2 evaluation board and the LSTBUS interface board

Once the code is loaded into the microcontroller start a HyperTerminal Window with the following settings:

- Bits per second: 115200
- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow Control: None
- Once connected, a carriage return will initiate the command line interface.
- To see a list of commands type a “?” in the terminal window.

List of MPL3115A2 commands:

```
Mn          : Mode 0=Standby; 1=Active;2=Alt;3=Bar
CC          : Check Status/Control Registers
CW          : Check Alarm Registers
RR xx       : Register xx Read
RW xx = nn  : Register xx Write value nn
RO n        : OSR Ratio  0=1;1=2;2=4;3=8;4=16;5=32;6=64;7=128
RT n        : ST Time Step  0=2^0;1=2^1;2=2^2;3=2^3;...;A=2^A;....F=2^F
RF          : Report OSR and Time Step and Mode
Sx          : Stream Data Polling
I n         : Stream via INTn
AAn xx yy wm wl : Altitude Alarm
APn xx yy wm wl : Pressure Alarm
ATn xx wm       : Temp and Window alarm
                :  n 1= INT1; 2= INT2
                :  xx: Target MSB; yy: Target LSB
                :  wm: Window MSB; wl: Window LSB
FO          : Stream via FIFO Overflow mode
FW ww       : Watermark Mode
            : ww: Watermark= 1 to 32
```

10.1 Modes

- M0 - Standby
- M1 - Active
- M2 - Altimeter
- M3 - Barometer

10.2 Register commands

10.2.1 Check Status and Check Window registers

The CC command lists out the contents of the following registers:

- Status Register (0x00/0x06)
- CTRL_REG1 (0x26)
- CTRL_REG2 (0x27)
- CTRL_REG3 (0x28)
- CTRL_REG4 (0x29)
- CTRL_REG5 (0x2A)
- System Mode Register (0x11)
- Interrupt Source Register (0x12)

The CW command lists the contents of the following registers:

- Altitude/Pressure Target MSB (0x16)
- Altitude/Pressure Target LSB (0x17)
- Altitude/Pressure Window MSB (0x19)
- Altitude/Pressure Window MSB (0x1A)
- Temperature Target (0x18)
- Temperature Window (0x1B)

10.2.2 Register Read

The RR command either lists the content of all registers in tabular format or if specified the contents of a single register i.e. RR 12 will list the contents of the Interrupt Source Register

10.2.3 Register Write

The RW command writes a value to a specific register i.e. RW 29 = 00

10.2.4 OSR Ratio

The RO command changes the OSR ratio i.e. RO 7 will change the OSR ratio to 128.

10.2.5 ST (Time Step) Ratio

The RT command changes the acquisition time step i.e. RT 2 will change the ST to 4 seconds.

Table 7. Time Step

Power	Time in Seconds
2^0	0
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^A	1024
2^B	2048
2^C	4096
2^D	8192
2^E	16384
2^F	32768

10.3 Alarms

10.3.1 Altitude/Pressure Alarms

To setup a threshold alarm at 300 meters on INT1:

AA1 01 2C

To setup a threshold and window alarm at a target of 300 meters with a delta of 20 meters on INT2:

AA2 01 2C 00 14

10.3.2 Temperature Alarms

To setup a threshold alarm at 25°C on INT1:

AT1 19

To setup a threshold and window alarm at a target of 25°C with a delta of 5°C on INT2:

AT2 19 05

10.4 FIFO Commands

The FO command will stream data via FIFO Overflow mode. The FW xx command will stream data via FIFO Watermark mode with the watermark set to xx i.e. FW 05 will set the watermark to 5.

10.5 Related Documentation

The MPL3115A2 device features and operations are described in a variety of reference manuals, user guides, and application notes. To find the most-current versions of these documents:

1. Go to the Freescale homepage at:

   http://www.freescale.com/

2. In the Keyword search box at the top of the page, enter the device number MPL3115A2.
3. In the Refine Your Result pane on the left, click on the Documentation link.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

AN4519
Rev 0.1
08/2012