I2C Communication Description

Product: Digital Mass Flow Sensor (KPI-DMFS-1)

Inter-Integrated Circuit (or I2C) is a widely used data bus allowing multiple devices to share data over short distances. In addition to a shared ground, only two primary lines are needed, a data line (SDA), and a clock (SCL). One or multiple devices act as a Master to send both addresses and commands to Slave devices, which respond according to the commands. The user will need to provide an external device configured and programmed as a Master in order to communicate with the Digital Mass Flow Sensor.

For every command, the slave address must be written as the first seven bits, with either a 0 or 1 as the last bit, 0 being Write, 1 being Read. If it is an address plus the write bit, after receiving an acknowledge, the Master must then write an 8-bit command. If the address plus the read bit is written, after receiving an acknowledge, the slave will respond with data.

The KPI Flow Sensor responds to reads with 3 bytes (except the “Read Serial Command” – see page titled “Serial Number” for more info), in the following order:

1. **Most significant data byte**
2. **Least significant data byte**
3. **CRC byte**

The basic sequence of commands for initializing the unit, for flow reading, is as follows:

1. Select Gas
    a. Optionally, verify gas selection (I2C Read)
2. Select flow type (LBM, SLPM)
    a. Optionally, verify flow type selection (I2C Read)
    b. After gas and flow selection have been verified, if these settings are not expected to ever change, you can optionally write the Save Settings command, and skip straight to step 3 upon subsequent start-ups
3. Start flow conversion
4. Now able to continuously read flow through I2C Read commands (values must be converted using divisor conversion chart)

If, instead of flow, temperature is wanted:

1. Select temperature
    a. Optionally, verify temperature selection (I2C Read)
    b. After temperature selection has been verified, if this setting is not expected to ever change, you can optionally write the Save Settings command, and skip straight to step 2 upon subsequent start-ups
2. Start conversion

3. Now able to continuously read temperature through I2C Read commands (values must be converted using divisor conversion chart)

<u>I2C Slave Address and Commands</u>

Digital Mass Flow Sensor Slave Address	
0x10	Hex
0b0010000	Binary

I2C Write and Read Commands, with Slave Address	
0x20	I2C Write, Hex
0x21	I2C Read, Hex

Digital Mass Flow Sensor Slave Commands	
Command	Description
0x01	Select Flow – Standard Liters Per Minute (SLPM)
0x02	Select Flow – Pound Mass Per Minute (LBM)
0x03	Select Temperature – Celsius (C)
0x04	Select Gas – Air
0x05	Select Gas – Oxygen
0x06	Read Serial Number
0x11	Start Conversion
0x77	Save Settings

When writing commands 0x01 through 0x05, the Digital Mass Flow Sensor will respond with those same commands upon receiving an I2C Read, in order that the user may confirm that the device received the correct command, if the user so wishes.

Ex:

1. MASTER: 0x20 (I2C Write)
2. MASTER: 0x04 (Select Gas – Air)
3. MASTER: 0x21 (I2C Read)
4. SLAVE: 0x00 (First Data Byte)
5. SLAVE: 0x04 (Second Data byte)
6. SLAVE: 0xC4 (CRC BYTE)

Therefore, the two-byte data bytes would be 0x0004, or 0x04, the same as the written command for Select Gas – Air.

Conversion Values

Each reading setting comes back as a 16-bit decimal value, and must be converted to its actual value using a different divisor, as listed below.

Setting	Divisor
SLPM	100.0
LBM	10000.0
Temperature	100.0

So, for example, if reading standard liters per minute, and the value comes back as decimal 15,784 – after dividing by 100.0, this value becomes 157.84 (SLPM), which is the flow rate.

Example I2C Flow Reading Sequence:

1. MASTER: 0x20 (I2C Write Command)
2. MASTER: 0x04 (Select Gas – Air)
3. MASTER: 0x20 (I2C Write Command)
4. MASTER: 0x01 (Select Flow – Standard Liters Per Minutes)
5. MASTER: 0x20 (I2C Write Command)
6. MASTER: 0x11 (Start Conversion)
7. MASTER: 0x21 (I2C Read)
8. SLAVE: First byte (Ex: 0x3D, or 0b00111101)
9. SLAVE: Second byte (Ex: 0xA8, or 0b10101000)
10. SLAVE: CRC byte (Ex: 0x36, or 0b00110110)
11. MASTER: Combine first and second bytes (0x3DA8, or 0b0011110110101000), to get the decimal value (15,784) – calculate the CRC on this decimal value, and ensure that it matches the received CRC byte (decimal 54) - then convert the decimal value using the chart and your particular setting (15,784 / 100.0 = 157.84 SLPM)

Serial Number

Each flow sensor is serialized with a 10-digit number, therefore the “read serial number” command (0x06) is the one command that returns more than three bytes. Instead, it returns nine bytes (6 data bytes and 3 CRC bytes), in the following sequence:

B1, B2, CRC1, B3, B4, CRC2, B5, B6, CRC3

Where B1 is the most significant data byte, and B6 is the least significant byte, and each CRC corresponds with the preceding two data bytes (CRC1 is the CRC value for 0xB1B2). The data bytes must be combined in sequence to get the resulting decimal serial number value, e.g. 0xB1B2B3B4B5B6.

Example Serial Number return values:

0x00, 0x01, 0xB0, 0x37, 0xD8, 0x20, 0x8C, 0xD6, 0xB4

In this scenario, **0x00, 0x01, 0x37, 0xD8, 0x8C, 0xD6** are the data bytes and **0xB0, 0x20, 0xB4** are the CRC bytes. If the CRC and data all match, we then combine the data bytes:

0x000137D88CD6

Which is the following decimal value:

5231906006

Which is the serial number of the unit.

Cyclic Redundancy Check (CRC)

A cyclic redundancy check is an error-detecting code used to expose data loss and accidental changes in values during data transfer. A calculation is performed internally on the data that is to be sent out from the device, and the remainder result is sent out alongside the data as the CRC code. The user can then perform the same calculation and validate that their operation computes the same CRC as the one received. If the CRC values match, we can say with high confidence that no data loss has occurred. If the CRC values do not match, it can be assumed that some data loss occurred, and the data needs to be re-transmitted.

CRC Calculation Example Code (Python)

```
def calculate_crc(i2cData):
      POLYNOMIAL = 0x131 #P(x)=x^8+x^5+x^4+1 = 100110001
      data = []
      data.append((i2cData >> 8) & 0xFF)
      data.append(i2cData & 0xFF)
      crc = 0xFF
      #calculates 8-Bit checksum with given polynomial
      for x in range(0,2):
           crc ^= (data[x]);
           for y in range(8,0,-1):
                if(crc & 0x80):
                     crc = (crc << 1) ^ POLYNOMIAL
                else:
                     crc = (crc << 1)
      return crc;
```

Where i2cData is a 16-bit integer value.

Example inputs/returns:

Input:	15784
Return:	54
Input:	4
Return:	69