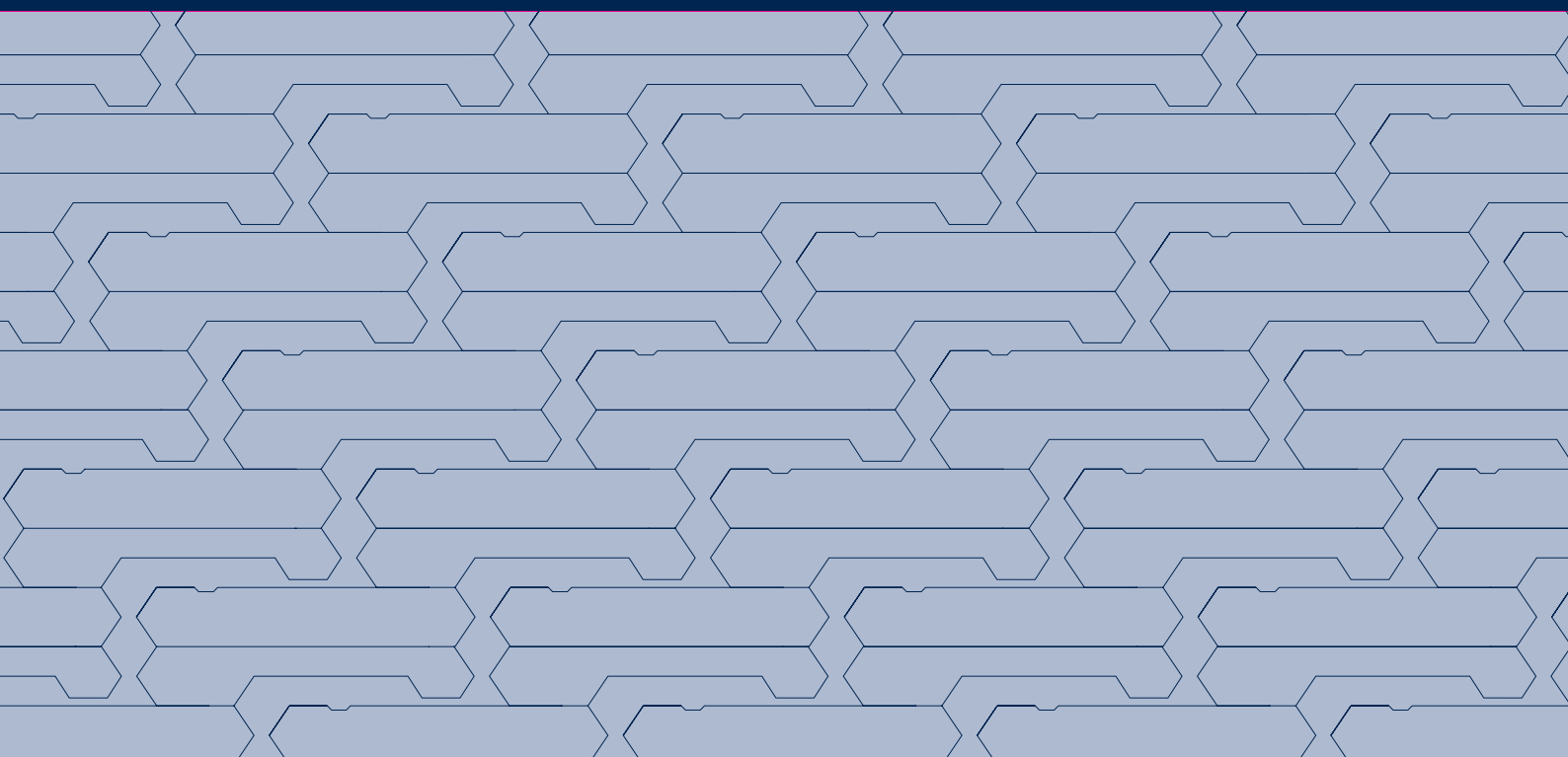
First Sensor
is now part of
TE connectivity

I²C bus communication with LMI pressure sensors

Application note



First Sensor is now part of TE connectivity AMSYS

I²C bus communication with LMI pressure sensors

This application note discusses the implementation of the Inter-Integrated Circuit (I²C) protocol used to communicate digitally with LMI series pressure sensors. Several typical application examples are also provided for different hardware platforms.

1. LMI pin description

Pin	Name	Function
1	-	Reserved
2	-	Reserved
3	GND	Supply ground
4	Vs	Supply voltage
5	ADR0	Hardware address 0
6	ADR1	Hardware address 1
7	SCL	Serial clock
8	SDA	Serial data

2. LMI connection diagram

Basic LMI connection diagram to an I²C host microcontroller is shown in Figure 1.

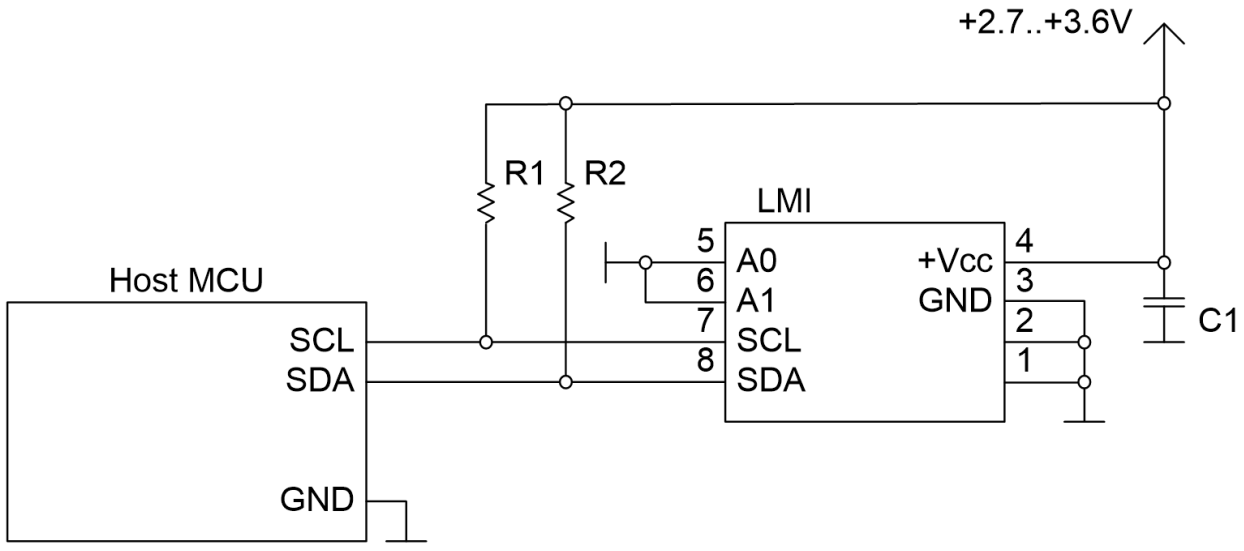


Fig. 1: Basic connection of an LMI sensor to a host microcontroller.

I²C bus communication with LMI pressure sensors

The sensor behaves as a slave on the I²C bus. Pins A0 and A1 define the sensor's address on the I²C bus as follows:

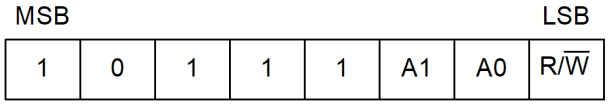


Fig. 2: I²C address composition of the LMI sensor.

Using 8-bit address notation, the base address of the sensor is 0xB8. The external address pins can be configured to modify the sensor's address according to Table 1.

A0	A1	7-bit address (hex)	8-bit address (hex)
0	0	0x5C	0xB8
1	0	0x5D	0xBA
0	1	0x5E	0xBC
1	1	0x5F	0xBE

Table 1. 7-bit and 8-bit address notations for different address settings.

The basic write sequence for LMI sensor is presented in Figure 3.

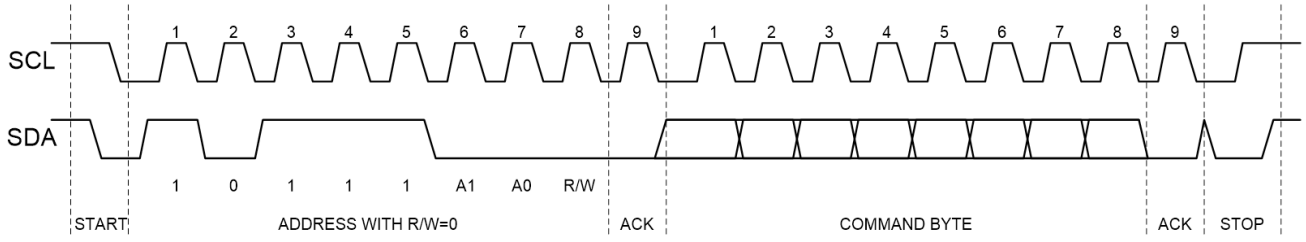


Fig. 3: Write command sequence

The I²C application interface for LMI sensor consists of the following commands:

Command name	Command byte (hex)	Description
Reset	0x11	Resets the sensor firmware
Blocking read	0x20	Starts conversion on data read
Start conversion	0x21	Starts conversion immediately

Table 2. Application command set

First Sensor is now part of TE connectivity

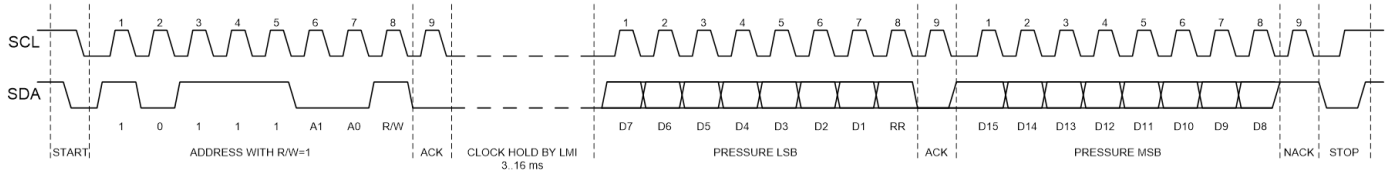
I²C bus communication with LMI pressure sensors

Minimum implementation requires only one of the two mutually-exclusive commands: **blocking read** or **start conversion** (non-blocking read). The firmware reset command is typically not required during normal sensor operation.

Blocking read mode

After receiving a **blocking read** command (0x20), the sensor doesn't start the conversion immediately. Instead, the conversion is executed each time the host sends the sensor's address with R/W bit set to read (0xB9, A0=A1=0). Re-sending the **start conversion** command is not required; the sensor will make a conversion upon each read.

The LMI sensor uses **clock stretching** in order to postpone the data readout until the pressure conversion is complete. During the conversion, the SCL line is held low by the LMI sensor. Once the data is ready, the clock is released and normal I²C operation resumes.



First conversion takes approximately 16 ms to complete. All of the following conversions will take approximately 5 ms, provided that the data is read and a subsequent request is made within 1.5 ms from the moment the sensor releases the SCL. Any delay of more than 1.5 ms sends the LMI sensor into sleep mode. A conversion following the sleep mode will take again approximately 16 ms.

If the host repeats reading the sensor continuously, the read cycle will have period defined by the LMI internal conversion logic, which is approximately 5 ms.

The host may read from 1 to 6 bytes of data from the sensor. To end the transmission, the host must submit NACK in the acknowledgement slot of the last byte.

The bytes in the data are the following:

Byte #	Bit #								Description
	1	2	3	4	5	6	7	8	
1	D7	D6	D5	D4	D3	D2	D1	RR	Least Significant Byte of pressure + RR
2	D15	D14	D13	D12	D11	D10	D9	D8	Most Significant Byte of pressure
3	T7	T6	T5	T4	T3	T2	T1	T0	Least Significant Byte of temperature
4	T15	T14	T13	T12	T11	T10	T9	T8	Most Significant Byte of temperature
5	V7	V6	V5	V4	V3	V2	V1	V0	Least Significant Byte of supply voltage
6	V15	V14	V13	V12	V11	V10	V9	V8	Most Significant Byte of supply voltage

Table 3. Data returned by blocking/non-blocking reads

RR bit in the LMI output is reserved, it's recommended to replace it with 0.
Bits D1..D15 contain signed (two's complement) conversion result (pressure).
Bits T0..T15 contain signed (two's complement) uncalibrated temperature measured by the sensor.
Bits V0..V15 contain signed (two's complement) uncalibrated supply voltage measured by the sensor.

I²C bus communication with LMI pressure sensors

Non-blocking read mode

After receiving a **start conversion** command, the sensor starts the pressure conversion immediately. The I²C bus becomes available for the host within a "window" of 1.5 ms after the conversion start was issued. For example, the host may start another LMI sensor which shares the same I²C bus.

After the conversion, the host may read the result. If the data is ready (i.e. the delay between the **start conversion** command and the read exceeds 17 ms), the read command will not block the bus and the data will be returned immediately. If the read attempt is done during the conversion (2..17 ms after the "Start conversion" command), the read would block the I²C bus and would return after the conversion is complete.

The read command returns the same data as in blocking read mode. Multiple subsequent reads would return the same value with the exception of the RR bit, which serves as "New data" flag.

RR=1 – new data
RR=0 – data previously read

To start a new conversion and receive new data, the user must send the start conversion command again.
This non-blocking mode is not suitable for continuous conversion because the conversion time will always be around 16 ms.

Clock stretching: hardware limitations

The LMI sensor uses a known I²C technique called "clock stretching" which is applicable to bytes and acknowledge bits, in particular to the blocking read bit. The sensor holds the SCL line low until the data is ready.

Most modern hardware supports I²C clock stretching, but not all of them (e.g. BCM2835/7 from Broadcom). In these cases, the user should avoid clock stretching by ensuring the following:

1) I²C clock speed does not exceed 100 kHz for all the devices on the bus
2) Host performs non-blocking read by doing the following sequence:
   - send "Start conversion" command
   - wait for 17 ms or more
   - read the data

This method is not suitable for continuous 5 ms read mode.

Clock stretching: software limitations

Standard MBED C/C++ API has a very short software timeout for most of the supported processors, which prevents clock stretching for more than a few milliseconds. This limitation can be bypassed by one of the following techniques:

- modifying the library to increase the timeout,
- using **start conversion** and non-blocking read, as described previously,
- using I2C::transfer C++ method or i2c_transfer_asynch C API function for asynchronous transfer,
- using directly processor-specific HAL driver API, free from this issue

I²C bus communication with LMI pressure sensors

Maximum clock speed

The maximum clock speed specified in the datasheet is 100 kHz, however, sensor operation has been tested with actual clock speeds up to 1 MHz. On the clock speeds below 100 kHz the sensor does not require clock stretching for each byte and acknowledgement bit. The sensor uses clock stretching to hold the bus for the blocking read only.

Please note that faster I²C clock and larger capacitance on I²C lines may require lower values for pull up resistors.

I2C bus start-up sequence

I²C protocol logic has a known issue: if a master is reset or reinitialized during read operation, the slave may get stuck holding the SDA bus low. If the master can be reset independently from slave, it would be wise to flush I²C bus each time before initializing the I²C peripheral module, to avoid possible bus hanging. This can be done by performing the following sequence:

1) Wait for a few milliseconds
2) Configure SDA in high impedance state,
3) Configure SCL as output (push-pull or open drain/open collector)
4) Clock 9 negative pulses on SCL while ignoring SDA

A practical implementation of this start-up sequence is shown in Figure 5.

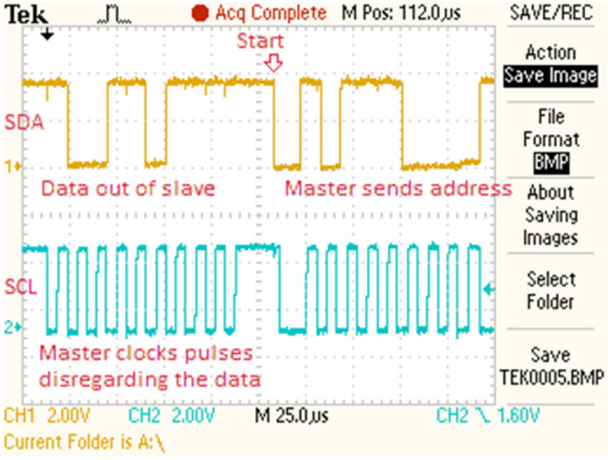


Fig. 5: Flushing the data out of slave on reset

If any of the slaves on the I²C bus has an unfinished data transfer in its buffer, it will be clocked out and the slave will release the bus.

For more on this issue, please see the references in the end of this document.

I²C bus communication with LMI pressure sensors

Reading LMI sensor with Arduino

LMI I²C lines are not 5V-tolerant, however, assuming that the I²C drivers are open-drain, the sensor can be used with 5V host processors providing that:

- the pull up resistors are connected to LMI power supply and
- the host processor's minimum high input level is lower than LMI power supply voltage.

For example, for Atmel's atmega328p processors the minimum high input level quoted in the datasheet is 0.6*Vdd = 3.0V, hence the sensor must be powered by at least 3V.

Using I²C with pull up resistors connected to +5V affects the data quality and must be avoided. Some of Arduino boards (e.g. Arduino Mega 2560) have on-board pull up resistors on hardware I²C, connected to +5V supply: this can be worked around by either:

1) Removing the on-board pull up resistors,
2) Adding external pull down resistors to counterweigh the on-board pull ups or
3) Using software I²C libraries on other pins

When using software I²C libraries, make sure to disable the internal pull up resistors.

The schematics of an LMI test bed with an Arduino Nano is given in Figure 6. Resistors R1-R2 must have value 1..22 kOhm depending on I2C clock speed as well as I²C bus length and capacitance.

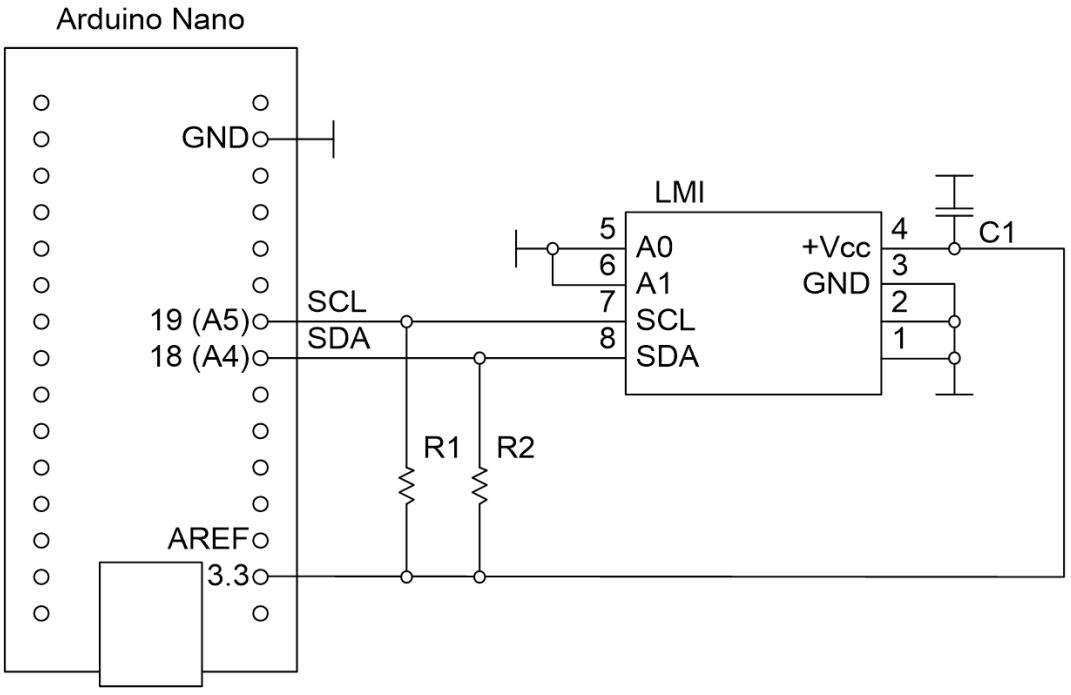


Fig. 6: LMI sensor connections to Arduino Nano I2C interface

I²C bus communication with LMI pressure sensors

Below is an example Arduino sketch which reads the pressure data and dumps the values into the serial interface.

```
#include <Wire.h>

#define BAUDRATE      115200          // Serial baud rate

#define I2C_ADDR      0xB8            // Default LMI address (A0=A1=0)
#define I2C_CLOCK     100000          // May also try 400000 or 1000000

#define CMD_GET_DATA  0x20            // “Blocking read” command

#define LED_PIN       13

void setup() {
  pinMode( LED_PIN, OUTPUT );         // Configure LED
  digitalWrite( LED_PIN, HIGH );      // Turn LED off

  Serial.begin( BAUDRATE );           // Configure serial output

  delay( 200 );                       // Wait 200 ms

  Wire.begin();                       // Configure I2C
  Wire.setClock( I2C_CLOCK );         // Set I2C clock rate

  Wire.beginTransmission( I2C_ADDR>>1 ); // Start I2C packet
  Wire.write( CMD_GET_DATA );         // Add command byte
  Wire.endTransmission();             // Finalize packet
}

void loop() {
  short data;

  digitalWrite( LED_PIN, LOW );       // Turn LED on
  Wire.requestFrom( I2C_ADDR>>1, 2 ); // Read two bytes from I2C
  digitalWrite( LED_PIN, HIGH );      // Turn LED off

  if( Wire.available() >= 2 ) {
    delay( 200 );                     // Wait 200 ms
    data = Wire.read();               // read low byte
    data |= ((short)Wire.read())<<8; // read high byte
    Serial.println( data );           // print the reading
  }
}
```

I²C bus communication with LMI pressure sensors

Reading LMI sensor with ARM using MBED API

The connection of LMI sensor to a generic board with STM32103C8T6 processor is given in Figure 7.

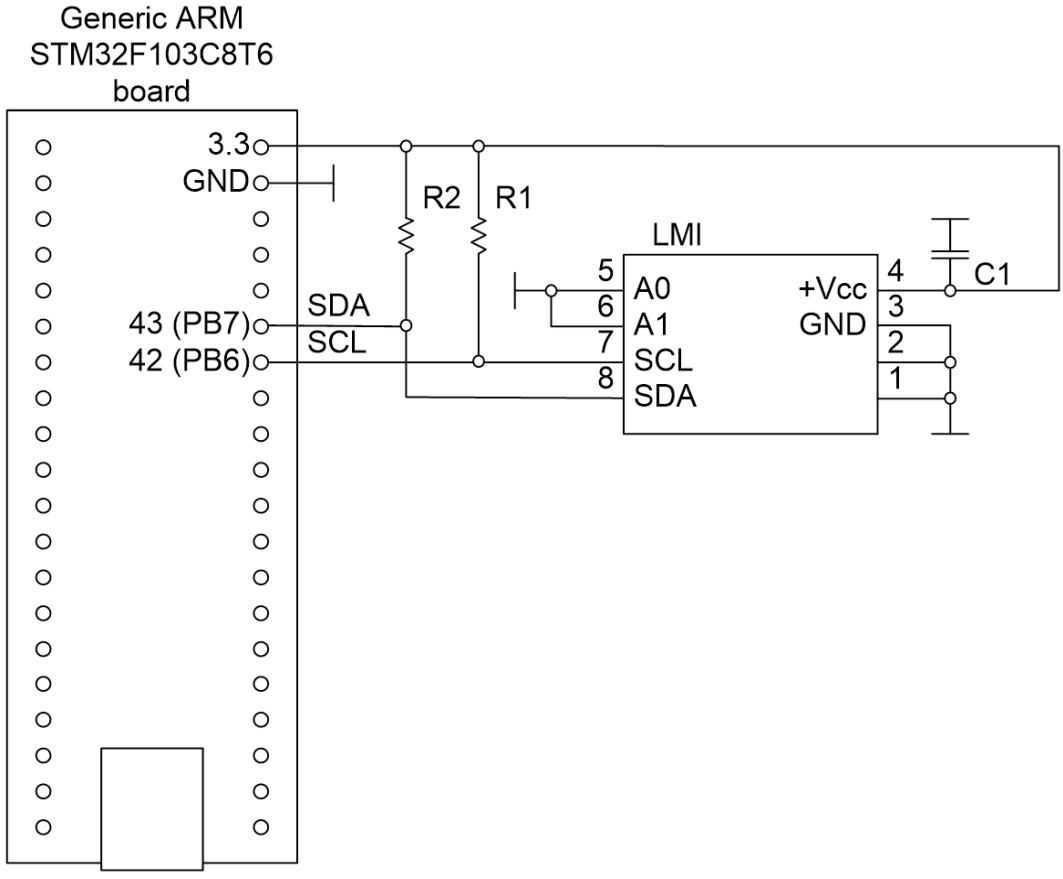


Fig. 7: The LMI sensor connection to generic "blue pill" board with STM32103C8T6 MCU

This board can be used with Arduino IDE (using Arduino Due compiler chain) In this case, the program example given in the previous section will work with minor modifications (namely, if you want to synchronize with LED, change the LED_PIN to PC13).

This board can also be programmed with MBED online/offline tools. See below for an example of reading an LMI sensor using MBED C++ I²C class.

I²C bus communication with LMI pressure sensors

```
#include "stm32f103c8t6.h"
#include "mbed.h"

#define BAUDRATE      115200         // Serial baud rate

#define I2C_ADDR      0xB8           // Default LMI address (A0=A1=0)
#define I2C_CLOCK     100000         // May also try 400000 or 1000000

#define CMD_CONV      0x21           // “Start conversion” command

#define SCL_PIN       PB_6
#define SDA_PIN       PB_7

#define SERIAL_TX     PA_2
#define SERIAL_RX     PA_3

#define LED_PIN       PC_13

int main()
{
  uint8_t byBuf[6];

  confSysClock();                     // Configure system clock
  Serial port( SERIAL_TX, SERIAL_RX ); // Configure serial port

  I2C i2c( SDA_PIN, SCL_PIN );        // Create I2C instance
  DigitalOut led( LED_PIN );          // Configure LED

  i2c.frequency( I2C_CLOCK );
  port.baud( BAUDRATE );

  while( true ) {
    wait_ms( 200 );
    led = 0;                                    // Turn LED on
    byBuf[0] = CMD_CONV;                        // Write command
    if( i2c.write( I2C_ADDR, (char*)byBuf, 1, true ) != 0 ) {
      port.printf( "Error writing command\r\n"); // Write error
    } else {
      wait_ms( 20 );              // Wait for the conversion completion
      if( i2c.read( I2C_ADDR, (char*)byBuf, 2, false ) != 0 ) {
        port.printf( "Error reading data\r\n");  // Read error
      } else {
        int ps = (byBuf[0] & 0xfe) | ((int)byBuf[1] << 8);
        port.printf( "PS = %d\r\n", ps );        // Post pressure value
      }
    }
    led = 1;                                    // Turn LED off
  }
}
```

First Sensor is now part of TE connectivity

I²C bus communication with LMI pressure sensors

Reading LMI sensor with ARM using MBED HAL drivers

See below for an example using MBED HAL drivers. The original program was written for and tested with STM32F411RE processor, I²C peripheral using pins PB_8 and PB_9.

```
#include "mbed.h"

#define BAUDRATE      115200         // Serial baud rate

#define I2C_ADDR      0xB8           // Default LMI address (A0=A1=0)
#define I2C_CLOCK     100000         // May also try 400000 or 1000000

#define CMD_GET_DATA  0x20           // “Blocking read” command

I2C_HandleTypeDef hi2c;

uint8_t byBuf[6]= {0};

// Private function prototypes
void I2C_Init(void);

int main(void)
{
// Reset all peripherals, initialize the Systick
  HAL_Init();

  Serial pc( USBTX, USBRX );
  pc.baud( BAUDRATE );

  printf( "Start\r\n" );

// Configure the system clock
//  SystemClock_Config();

// Initialize I2C peripherals
  I2C_Init();

  HAL_Delay( 200 );

// Send “Blocking read” command
  byBuf[0] = CMD_GET_DATA;
  if( HAL_I2C_Master_Transmit( &hi2c, I2C_ADDR, byBuf, 1, 500) != HAL_OK )
  {
    printf( "Error TX\r\n" );
//    Placeholder for error handler;
  }

// Reading loop
  while (1) {

// Wait until the bus is ready
    while( HAL_I2C_GetState( &hi2c ) != HAL_I2C_STATE_READY );
```

I²C bus communication with LMI pressure sensors

```
// Read 4 bytes from LMI sensor (pressure + uncalibrated temperature)
    if( HAL_I2C_Master_Receive( &hi2c, I2C_ADDR, byBuf, 4, 500) != HAL_OK )
    {
      printf( "Error RX\r\n" );
//      Placeholder for error handler;
    } else {
      printf( "PS = %hd, T = %hd\r\n",
        ((int16_t)byBuf[1] << 8) | byBuf[0],
        ((int16_t)byBuf[3] << 8) | byBuf[2] );
    }
    HAL_Delay( 200 );
  }
}

// GPIO init function (called from HAL_Init)
void HAL_I2C_MspInit( I2C_HandleTypeDef *hi2c )
{
  GPIO_InitTypeDef GPIO_InitStruct;

  __HAL_RCC_GPIOB_CLK_ENABLE();

  GPIO_InitStruct.Pin = GPIO_PIN_8;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FAST;
  GPIO_InitStruct.Alternate = GPIO_AF4_I2C1;
  HAL_GPIO_Init( GPIOB, &GPIO_InitStruct);

  GPIO_InitStruct.Pin = GPIO_PIN_9;
  HAL_GPIO_Init( GPIOB, &GPIO_InitStruct);

  __I2C1_CLK_ENABLE();
}

// I2C init function
void I2C_Init( void )
{
  hi2c.Instance = I2C1;
  hi2c.Init.ClockSpeed = I2C_CLOCK;
  hi2c.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c.Init.OwnAddress1 = 0;
  hi2c.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c.Init.OwnAddress2 = 0;
  hi2c.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;

  if( HAL_I2C_Init( &hi2c ) != HAL_OK )
  {
    printf( "Error Init\r\n" );
//    Placeholder for error handler;
  } else {
    printf( "Init OK\r\n" );
  }
}
```

I²C bus communication with LMI pressure sensors

References

1. I2C Tips, chapter "External Slave Device Hanging the Bus by Holding SDA Low"
2. The application note AN-686from Analog Devices.