# Open Source Barologger Tag

### Version 3.0.2, programmed with Energia 17

### Purpose

To provide a lightweight pressure, temperature, and humidity datalogger that is programmable using a simple, high-level language for the study of movement in small animals.

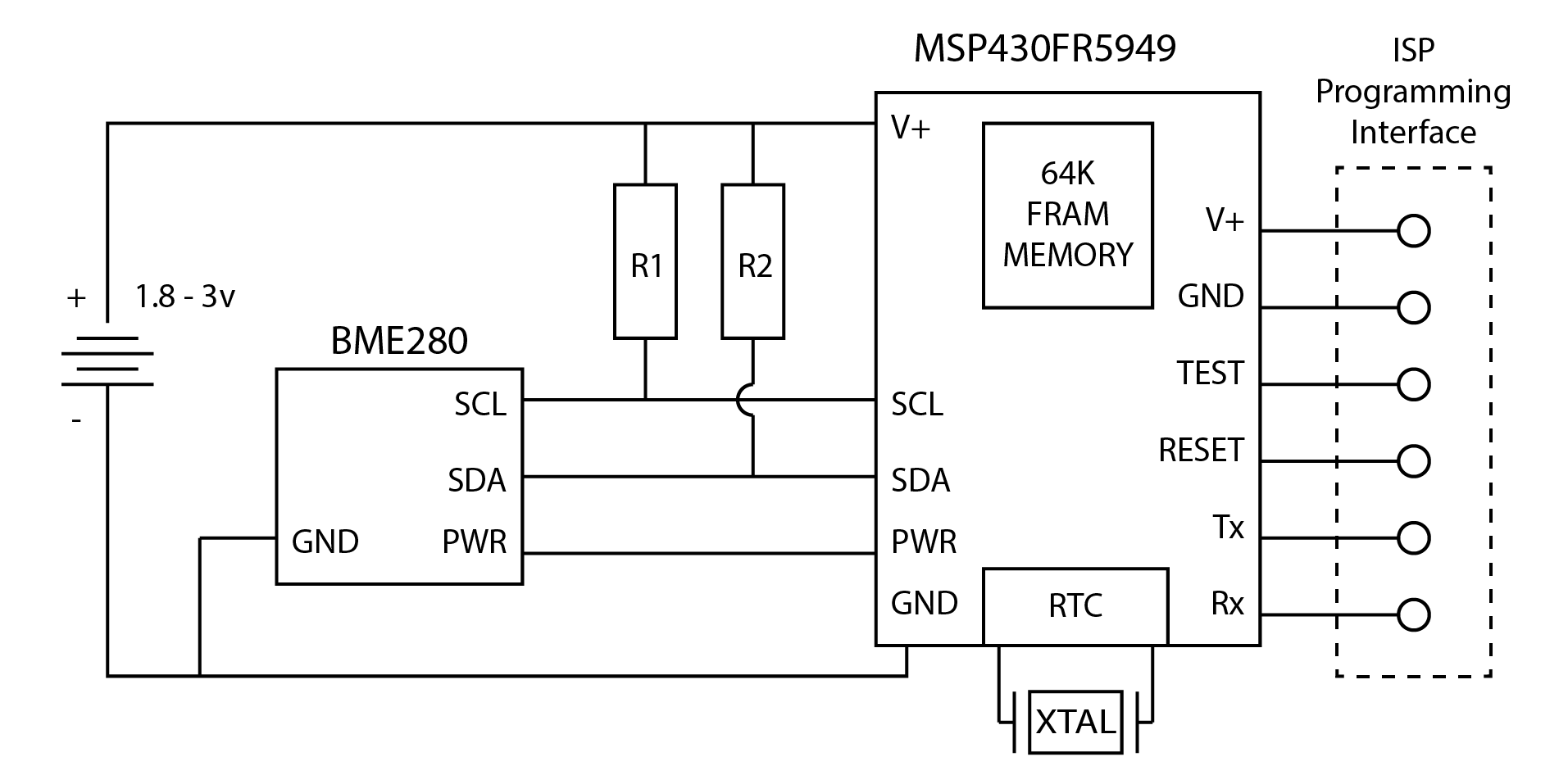
### Overview

The datalogger sensor polling, sampling rate, duty cycle, and memory allocation is managed by a microcontroller with onboard ferromagnetic memory (FRAM). FRAM has several advantages over other memory formats (e.g. FLASH, EEPROM, SRAM, etc) by using less power, is rewritable, and is non-volatile.

The pressure sensor is a microelectromechanical system (MEMS) that is controlled using an I2C interface.

The microcontroller minimizes power usage by powering down different systems when they are not necessary for operation. For example, when the datalogger isn’t gathering information from the sensor about the local environment, the only necessary operations are keeping track of the time between each sample. Thus, after gathering a sample and recording it to FRAM, the microcontroller places the sensor in sleep mode to minimize power usage and turns off other various peripheral functions of the microcontroller before placing itself in sleep mode until the next sampling.

To minimize drift and maintain accurate time intervals between samplings, the microcontroller uses an external 32.748 KHz crystal oscillator.



### Hardware

Texas Instruments MSP430FR5949

The MSP430™ ultra-low-power (ULP) FRAM platform combines uniquely embedded FRAM and a holistic ultra-low-power system architecture, allowing innovators to increase performance at lowered energy budgets. FRAM technology combines the speed, flexibility, and endurance of SRAM with the stability and reliability of flash at much lower power.

The MSP430 ULP FRAM portfolio consists of a diverse set of devices featuring FRAM, the ULP 16-bit MSP430 CPU, and intelligent peripherals targeted for various applications. The ULP architecture showcases seven low-power modes, optimized to achieve extended battery life in energy-challenged applications.

The complete datasheet can be found [here](http://www.ti.com/lit/gpn/MSP430FR5949).

Bosch BME280

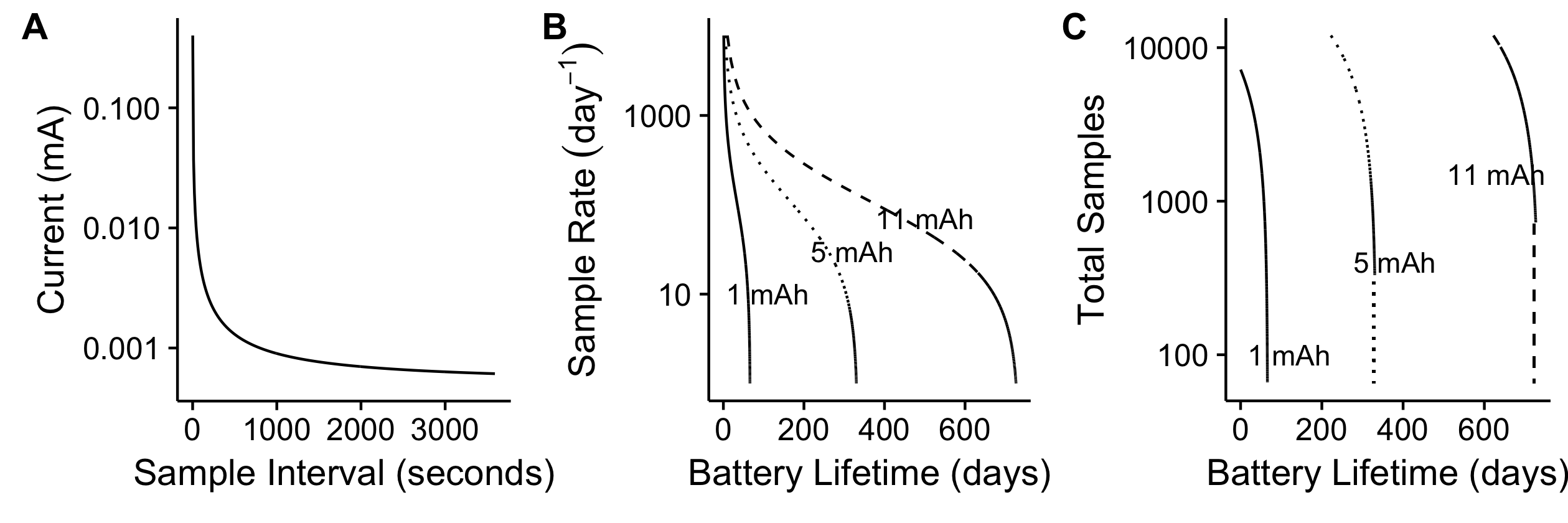
The BME280 is an integrated environmental sensor developed specifically for mobile applications where size and low power consumption are key design constraints. The unit combines individual high linearity, high accuracy sensors for pressure, humidity and temperature in an 8-pin metal-lid 2.5 x 2.5 x 0.93 mm³ LGA package, designed for low current consumption (3.6 μA @1Hz), long term stability and high EMC robustness.

The humidity sensor features an extremely fast response time which supports performance requirements for emerging applications such as context awareness, and high accuracy over a wide temperature range. The pressure sensor is an absolute barometric pressure sensor with features exceptionally high accuracy and resolution at very low noise. The integrated temperature sensor has been optimized for very low noise and high resolution. It is primarily used for temperature compensation of the pressure and humidity sensors, and can also be used for estimating ambient temperature.

The BME280 supports a full suite of operating modes which provides the flexibility to optimize the device for power consumption, resolution and filter performance.

The complete datasheet can be found [here](https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BME280_DS001-11.pdf).

### Power Consumption and Duty Cycle



### Previous Versions

The following table is a list of components that were used in previous versions of the datalogger.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Hardware Version | | |
| Component | *Type* | 1.0 | 2.0 | 3.0 |
| MSP430FR5969 | Microcontroller | • | • |  |
| MSP430FR5949 | Microcontroller |  |  | • |
|  |  |  |  |  |
| BMP180 | Sensor | • |  |  |
| BMP280 | Sensor |  | • |  |
| BME280 | Sensor |  |  | • |

### Libraries



The datalogger uses the following libraries to communicate with the sensor using I2C, set the appropriate registers and settings for the sensor, and to keep track of time.

BME280\_MOD-1022.h links to a file

Wire.h is the default I2C library provided with Energia

RTClib.h is a library to keep track of time in common units

All library files should be placed within the Energia folder specifically for libraries. This is typically placed in the ~/User/Documents/ folder during installation as the folder ~/User/Documents/Energia/. More information detailing installation of libraries and general Energia troubleshooting can be found at the following link -

<http://energia.nu/guide/library-manager/>

### Demo Code



To write data into the non-volatile FRAM, variables are pre allocated into the section(“.text”) in the available memory. In the above code snippet, the number 5000 in the brackets is the number of samples. There is a limit of 64k memory including the program overhead.



The setup loop “void setup()” runs once and initializes the libraries that will be used by the microcontroller.

clockSetup(); is a local function that sets the registers for the external 32.768 kHz crystal. More data is provided below on the individual parts of the function.

setPins(); is a local function that sets the unused pins of the microcontroller to LOW to prevent pin floating and reduce power consumption during low power mode.

Wire.begin() initializes“Wire”, the library we will use to communicate with the sensor via I2C.

Serial.begin() initializes “Serial”, the library that is used for communications between the computer and the datalogger. Serial is only kept initialized during testing, and is typically used to ensure the microcontroller is reading data from the sensor correctly.

downloadData(); is a local function that provides readout of data stored on the FRAM. It has a default timeout of 15 seconds before it exits the loop and begins logging data again.

pinMode(BME280\_power, OUTPUT) is setting the status of the power pin for the Bosch BME280 sensor variable, “BME280\_power” as an output pin.

rtc.begin(DateTime(F(\_DATE\_), F(\_TIME))); is from the RTC library, a function that captures local time from the PC that is programming the datalogger.

rtc.adjust(DateTime(…); is from the RTC library, a function that can adjust the set local time if there is a delay due to the compiler. Default is set at 28 seconds.

timestamp(); is a local function that records the compile time to FRAM.



The “void loop()” repeats indefinitely until the memory is full or the power source drops below operational voltage. It controls the basic functions of recording the sensor data and low power modes.

wakeUp() is a function that wakes the sensor from sleep mode.

BME280Settings() controls the basic settings of the sensor including operating mode and oversampling rate. The functions contained are provided in more detail below.

BME280.readCompensationParams() is a function that provides the onboard compensation parameters from the factory.

BME280.readMeasurements() is a function that reads the measurements from the sensor. The measurements are read in a specific order; humidity, temperature and pressure to ensure accuracy.

serialTestMode() is a function that provides serial output via terminal for troubleshooting.

gotoSleep() is a local function that sets the datalogger into low power mode.



downloadData() is a local function that provides readout of data stored on the FRAM. It has a default timeout of 15 seconds before it exits the loop and begins logging data again. Pressing any key and hitting “send” on terminal or serial monitor will enter the download loop. The variable address count increases each turn until it reaches the maximum number of samples (default = 5000) and reads out the variables stored in the FRAM across the serial terminal.



clockSetup() is a local function that sets the main clock to use the external 32.768 kHz crystal and clear faults until the oscillator is stable.



setPins() function controls the default status of the unused pins of the microcontroller. All except the Serial pins (28 and 29), I2C pins (17 and 18), and the sensor power pin (pin 1) are set LOW.



BME280Settings() function controls the basic settings of the sensor including operating mode and oversampling rate.

BME280.writeMode()

BME280.writeOversamplingPressure() controls the oversampling rate of the sensor when measuring air pressure. There are 5 settings available –

os1x is a single sample, os2x averages 2 samples, os4x averages 4 samples, os8x averages 8 samples, and os16x averages 16 samples

The oversampling rate illustrates a tradeoff between accuracy and power consumption, as os16x provides the highest sensor accuracy but is using power 16 times as long as os1x.



gotoSleep() is a local function that places the datalogger into low power mode.



deployMode() is a local function that reads the data from the sensor and writes it to the FRAM. Sample count increases each turn until the reaches the maximum set (5000 in default).

### Programming

The device can be programmed using the appropriate version of Energia listed at the top of this document. It is programmed using Inline Serial Programming (ISP) protocol coupled with either a homemade connector or the provided programmer template.

The programmer can be connected to the MSP430FR5969 Launchpad by removing the following jumpers that connect between the debugger and the target MSP430. This disables the target MCU and makes the programmer board the target.

<http://43oh.com/2011/11/tutorial-to-use-your-launchpad-as-a-programmer/>

We are using 6 total lines to communicate with the target datalogger using ISP, including 2 to ensure the datalogger is functioning properly (Rx and Tx for Serial communication).