ISM400 Sensor Board

Advanced User's Guide

August, 2010

ILLINOIS STRUCTURAL HEALTH MONITORING PROJECT
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
OPEN SYSTEMS LABORATORY & SMART STRUCTURES TECHNOLOGY LABORATORY

1 Introduction

The purpose of this *Guide* is to provide detailed instructions on using the ISM400 (formerly SHM-A) sensor board. In particular the guide will allow you to:

- Create custom configurations with user-selected sampling rates and filter designs
- Obtain and apply temperature correction coefficients for individual sensor boards

This guide assumes that you are familiar with the basic use of the ISM400 sensor board for data acquisition using the instructions provided in the *ISHMP Structural Health Monitoring User's Guide*.

2 Creating a Channel Configuration File

The sensing software provided in the ISHMP Services Toolsuite has four default configuration files (see Table 1). The channel configuration file supplies the following information for the QF4A512 chip (the programmable signal conditioner at the heart of the ISM400 sensor board):

1. Active channels (any combination of channels 1, 2, 3 and 4)
2. Programmable Gain Amplifier (PGA) setting (1x, 2x, 4x, 8x) for each channel
3. Sampling rate for each channel
4. FIR filter coefficients for each channel
5. SPI (digital bus) clock rate

The default configuration files included in the ISM400 (formerly SHM-A) driver (shm/sensorboards/SHM_A) have the following properties:

Table 1. Properties of default configuration files

Sample Rate (Hz)	Digital Cut-off Frequency (Hz)	Active Channels	Gain	Latency (points)	File Name
10	4	1,2,3	1	76	filter3ch_fs10Hz_fc4Hz.h
50	20	1,2,3	1	89	filter3ch_fs50Hz_fc20Hz.h
100	40	1,2,3	1	78	filter3ch_fs100Hz_fc40Hz.h
280	70	1,2,3	1	94	filter3ch_fs280Hz_fc70Hz.h

If a user requires different sampling rates or different active channels than the ones provided, a custom configuration file may be created and implemented in the ISM400 sensor board driver prior to compilation of the sensing application.

Channel configuration files are created using the free Quickfilter Pro Software which may be downloaded from: http://www.quickfiltertech.com/html/content_page.php?content_id=33&. Once the software is successfully installed, open the program by double-clicking on the QF4A512Config Icon. When the program opens you will get an error message that reads: "Fail to connection to Evaluation board." This is not a problem so you may click OK to display the QF4A512Config window. The following instructions will guide you through using this software to create a new configuration and implementing it in the sensing application.

2.1 Determine channel parameters

Prior to creating a new channel configuration the necessary channel parameters must be identified. The channel parameters that are required for the configuration file are:

- Number of active channels (up to 4)
- Sampling rate (6 to 5000 Hz)
- Gain (1x, 2x, 4x, 8x)
- Filter characteristics (type, cut-off frequency, roll-off, ripple, etc.)

2.2 Design FIR filter

The sampling rate and filter characteristics implemented on each channel are defined by the FIR filter that is chosen for that channel. The QF4A512 Filter Editor software is used to create the desired FIR filter(s). The Filter Editor may be accessed by double-clicking on one of the empty boxes below the **Filter** heading along the top of the block diagram in the top half of the main window as shown in Figure 1.

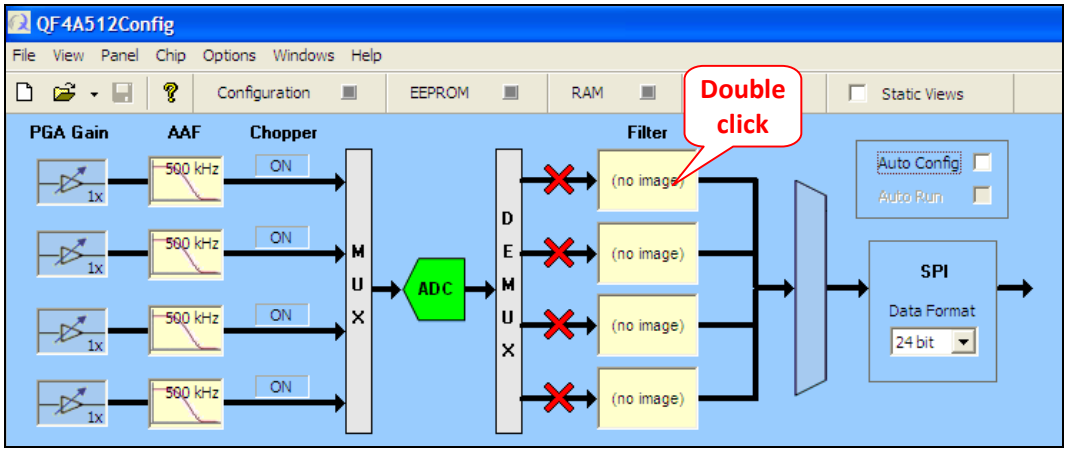

Figure 1. Configuration window (top half) of QF4A512 Filter Editor software

This will open the Filter Editor displaying several options for filter type as shown in Figure 2. For basic sensing applications a Basic McClellan Parks Lowpass filter is suggested.

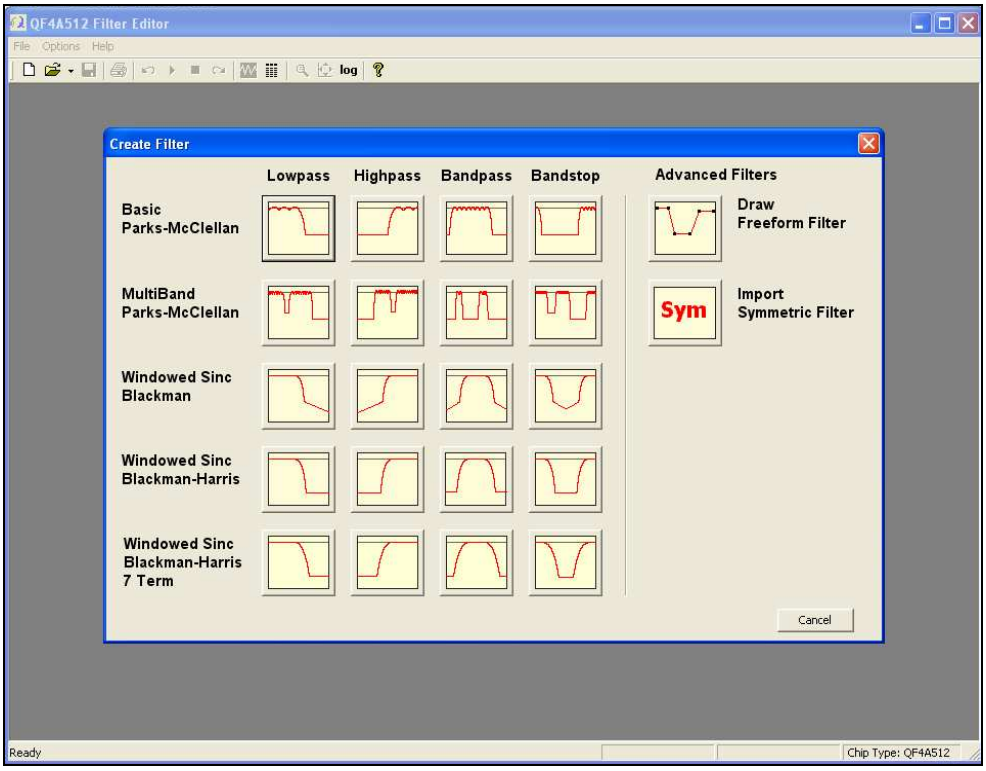

Figure 2. Opening Filter Editor

Clicking on the button corresponding to the desired filter type will bring up the FIR Specification Editor for that specific filter. Figure 3 shows the editor for the Basic Parks-McClellan Lowpass filter. Notice that the first input field defines the sampling rate. The filter cutoff frequency must be less than the sampling rate/2.2. Once all of the fields are filled out click on "`Design Filter`". If you have chosen sampling rate/filter parameters that are not supported you will receive an error message with suggested corrections to the design. Once the filter has been successfully designed, save it by going to `File` → `Save Filter As`. This filter file will be retrieved later when assigning filters to each channel in your configuration. Close the Filter Editor to return to the configuration window.

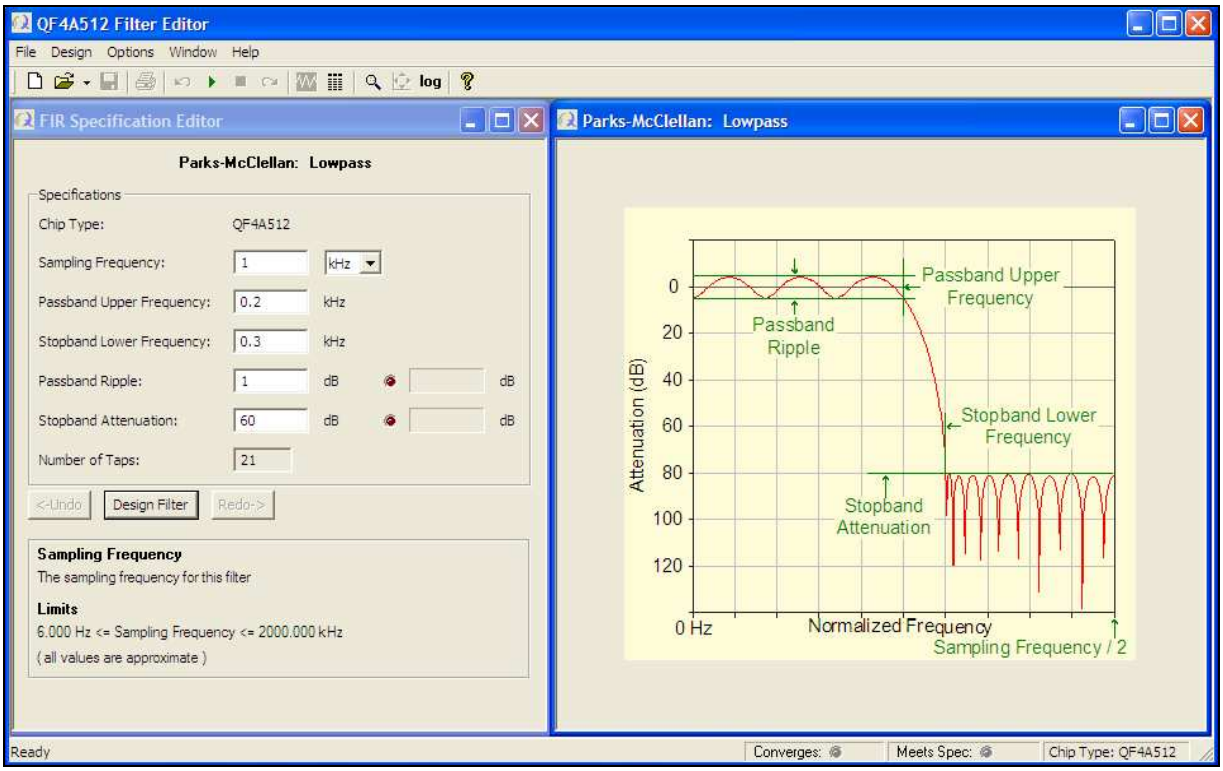

Figure 3. Editor for the Basic Parks-McClellan Lowpass filter

2.3 Assign filter and parameters to each channel

On the bottom half of the configuration window, click on the right arrow next to the Configuration Results heading as shown in Figure 4. This will take you to the parameters for Channel 1 (see Figure 5).

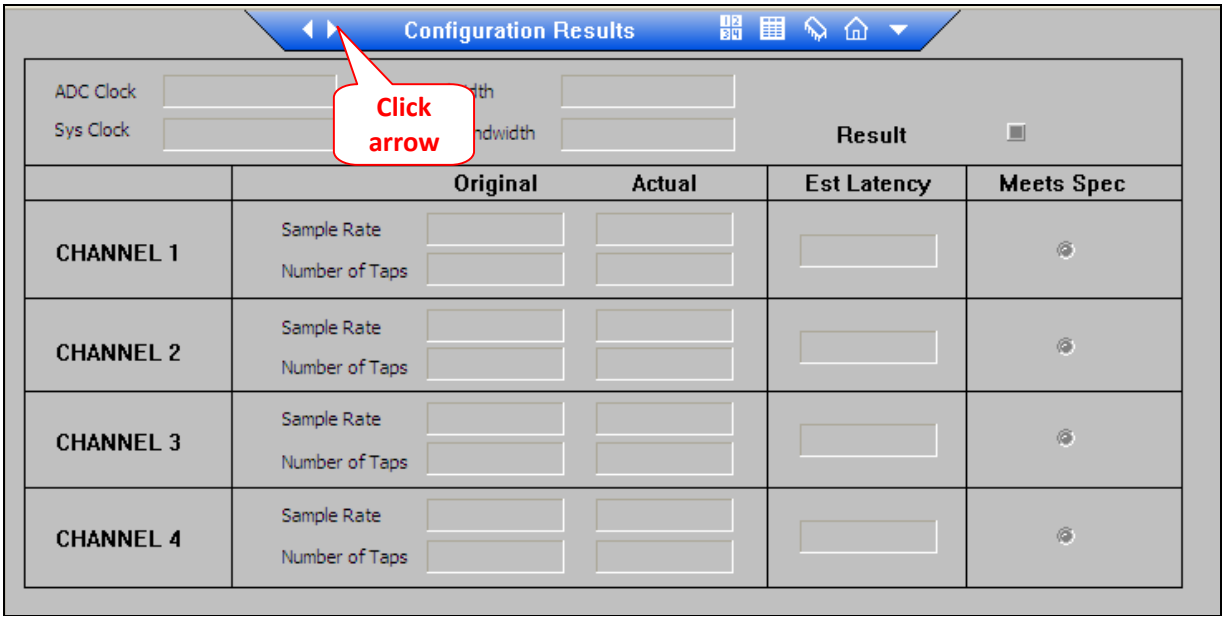

Figure 4. Configuration window (bottom half)

In the Filter Attributes window, click on Load. Select the filter that you created for Channel 1 as shown Figure 5.

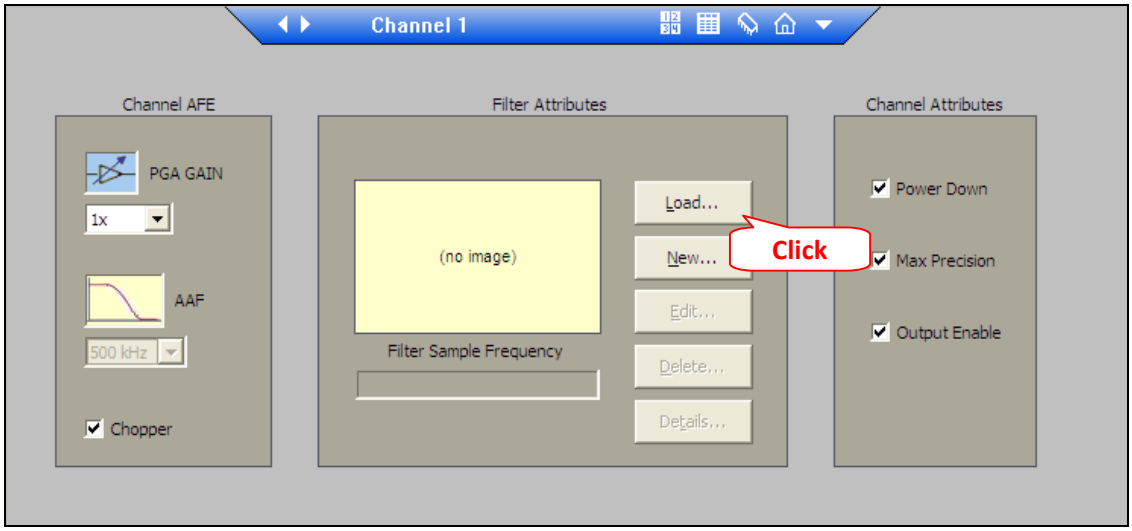

Figure 5. Filter attributes window

The selected filter will be displayed in the Filter Attributes box as well as in the overall block diagram in top half of the configuration window. Use the arrows in the blue tab to scroll through all of the channels and assign filters for all of the channels that you plan to have active.

If a gain other than unity is desired for any of the channels it can be set by selecting the PGA GAIN in the dropdown menu in the Channel AFE portion of the window. Note that a gain other than 1x runs the possibility of reaching the ADC limit and producing clipped data.

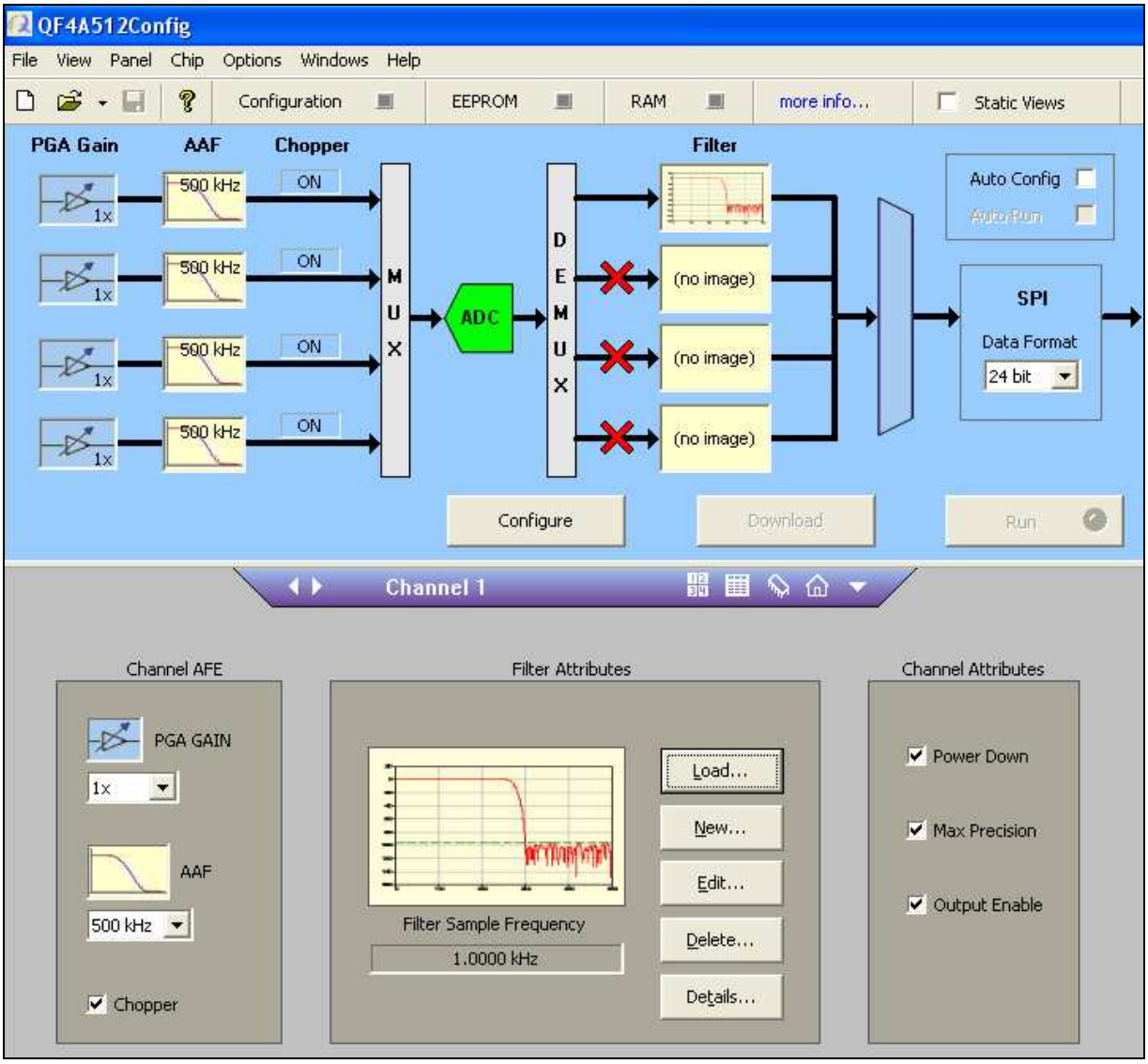

Figure 6. Configuration window (after setting up channel 1)

2.4 Set Clock Settings

Continue to scroll through the configuration options using the arrows in the blue tab until you reach Clock Settings as shown in Figure 7. Under System Clocking, in the Host SPI Bandwidth field, set the SPI rate to 13 MHz (Imote2 SPI rate). Ensure that the external clock rate is set to 20MHz as shown below.

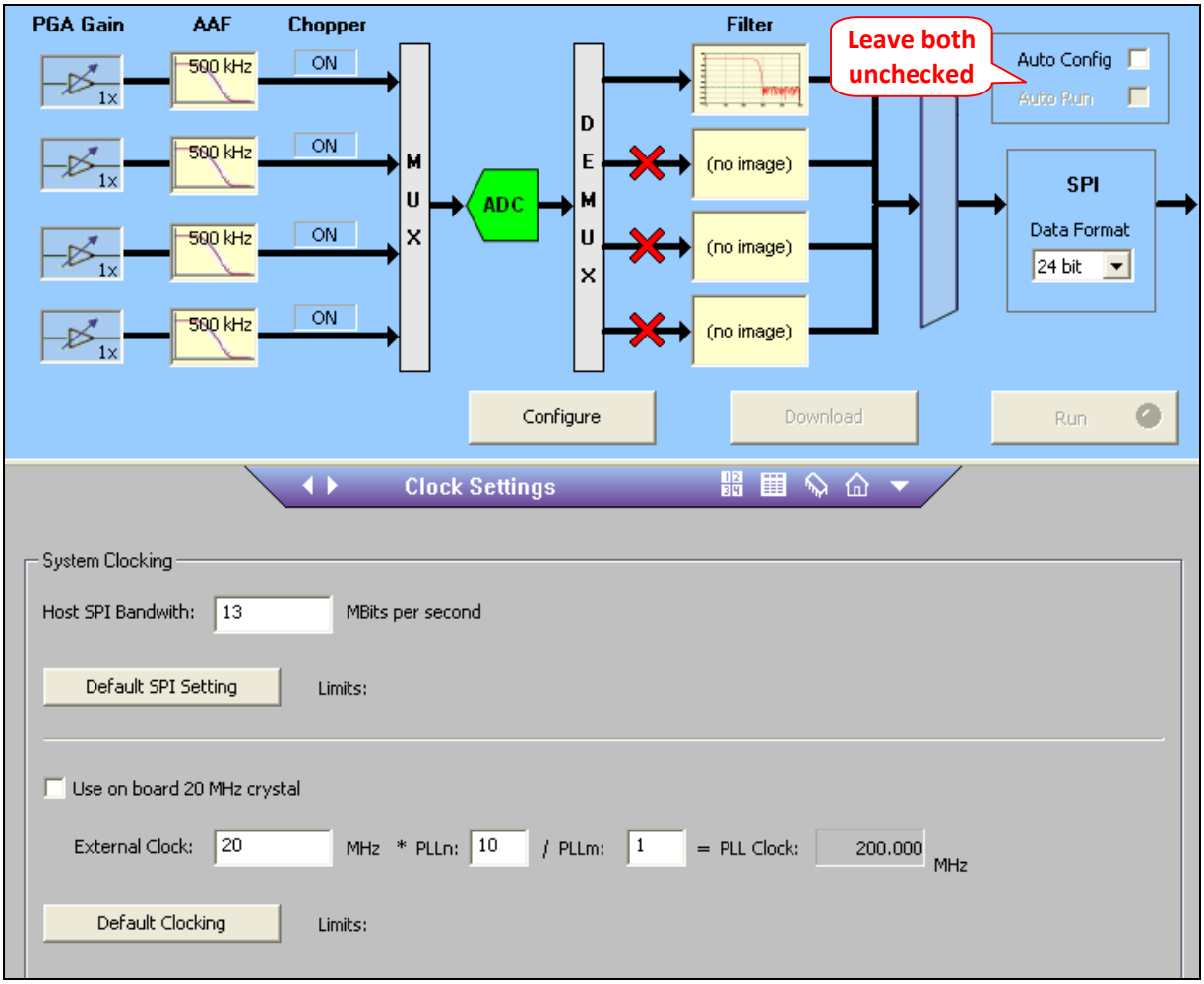

Figure 7. Window for clock settings

In the box in the top right of the window, ensure that `Autoconfig` and `Autorun` are NOT checked.

2.5 Save the Configuration

Click the Configure button in the middle of the screen. Upon successful configuration (indicated by the log created in the bottom right portion of the window) save the configuration by going to File → `Save Configuration As`. This saves the configuration as a .qfp file but TinyOS requires a header (.h) file. To create a header file go to File → `Export Image`. This will bring up and Export Configuration window. Check the `Export from Configuration File` option and then click `Browse` to locate the desired configuration file. Under `Export Files` check the C code option to create a `.h` file that can be directly included with the TinyOS driver. Specify the location of the saved file (somewhere it can easily be found) and then click `Export` to save the file to the specified location (see Figure 8).

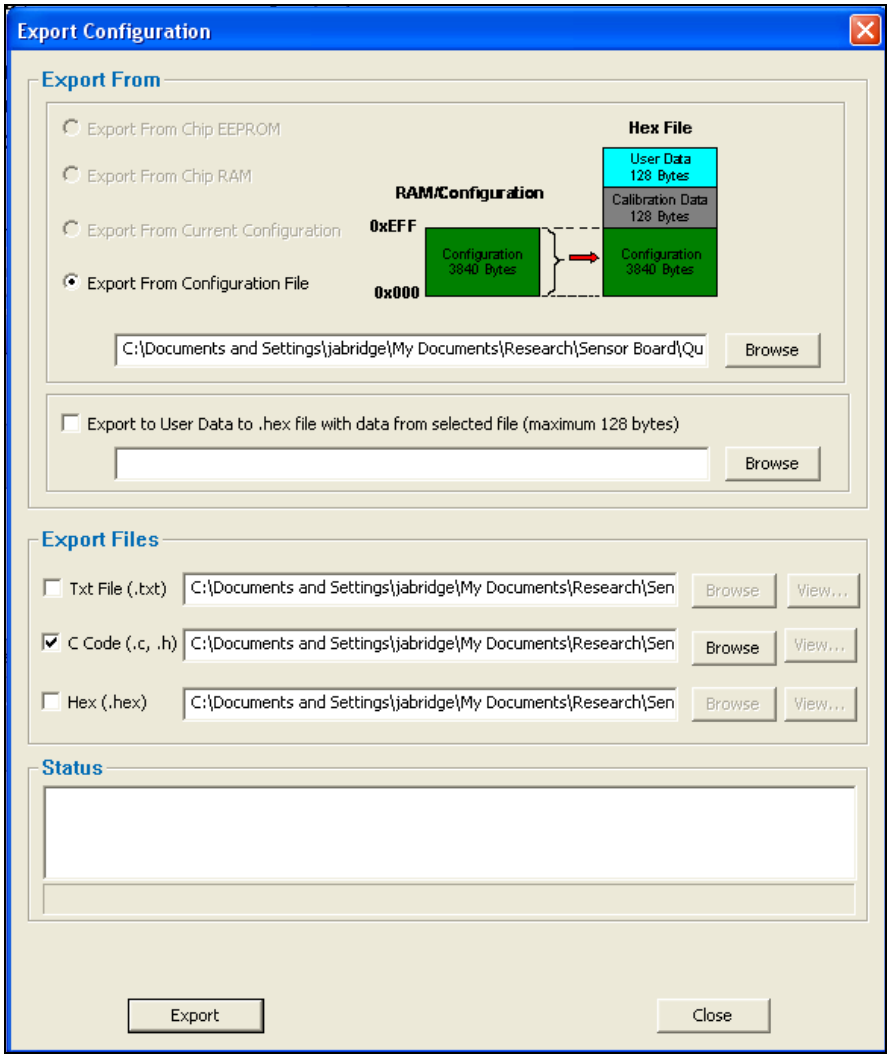

Figure 8. Saving configurations

2.6 Update the ISM400 driver files

Copy the .h file created in the last step to shm/sensorboards/SHM_A directory and rename `filterCustom.h`. If a previous `filterCustom.h` exists in the SHM_A directory, just overwrite it.

1. Update driver files

The channel configuration file is included in the sensing application. Four default files have been created. When the sensing application is called, the user-specified sampling rate determines which configuration file is used. Table 1 summarizes the configuration files. Note that the configuration file saved as `filterCustom.h` may be used for a user-defined configuration.

Once the `filterCustom.h` file is created, the sampling rate defined by users should be registered in the driver code named `filter.h` to use the sampling rate. Open the `filter.h` file (shm/sensorboards/SHM_A/filter.h), and change the custom filter rate as shown in Figure 9.

```
90 #ifndef _FILTERS_H
91 #define _FILTERS_H
92
93 #include "filterCustom.h" /* must be included first - defines QFRegister */
94 #include "filter3ch_fs10Hz_fc4Hz.h"
95 #include "filter3ch_fs50Hz_fc20Hz.h"
96 #include "filter3ch_fs100Hz_fc40Hz.h"
97 #include "filter3ch_fs280Hz_fc70Hz.h"
98 #include "calibrationFilter.h"
99
100 #ifndef SHMA_CUSTOM_FILTER_RATE
101 #define SHMA_CUSTOM_FILTER_RATE (1000)
102 #endif
103
104 const int *QFImageRegisterTableSizeArray[] = {
105                                               &QFImageRegisterTableSize10,
106                                               &QFImageRegisterTableSize50,
107                                               &QFImageRegisterTableSize100,
108                                               &QFImageRegisterTableSize280,
109                                               &QFImageRegisterTableSize };
110
111 const struct QFRegister *QFImageRegisterTableArray[] = {
112                                               QFImageRegisterTable10,
113                                               QFImageRegisterTable50,
114                                               QFImageRegisterTable100,
115                                               QFImageRegisterTable280,
116                                               QFImageRegisterTable };
117
118 const uint32_t QFDelayArray[] = {
119                                    80,//for 10Hz config
120                                    89,//for 50Hz config
121                                    78,//for 100Hz config
122                                    94,//for 280Hz config
123                                    83 };//for custom filter config
124
125 #endif
```

Figure 9. Registration of user-defined sampling rate (e.g. 1000Hz)

The FIR filter causes delay in the digital output of the data. To allow the synchronization of the data from the ISM400 sensor board and other sensor boards, the latency must be included in the `filters.h` file. The delay constant is defined in the number of data points and can be determined as follows:

1. Note the latency given in QFConfig after the configuration is completed.

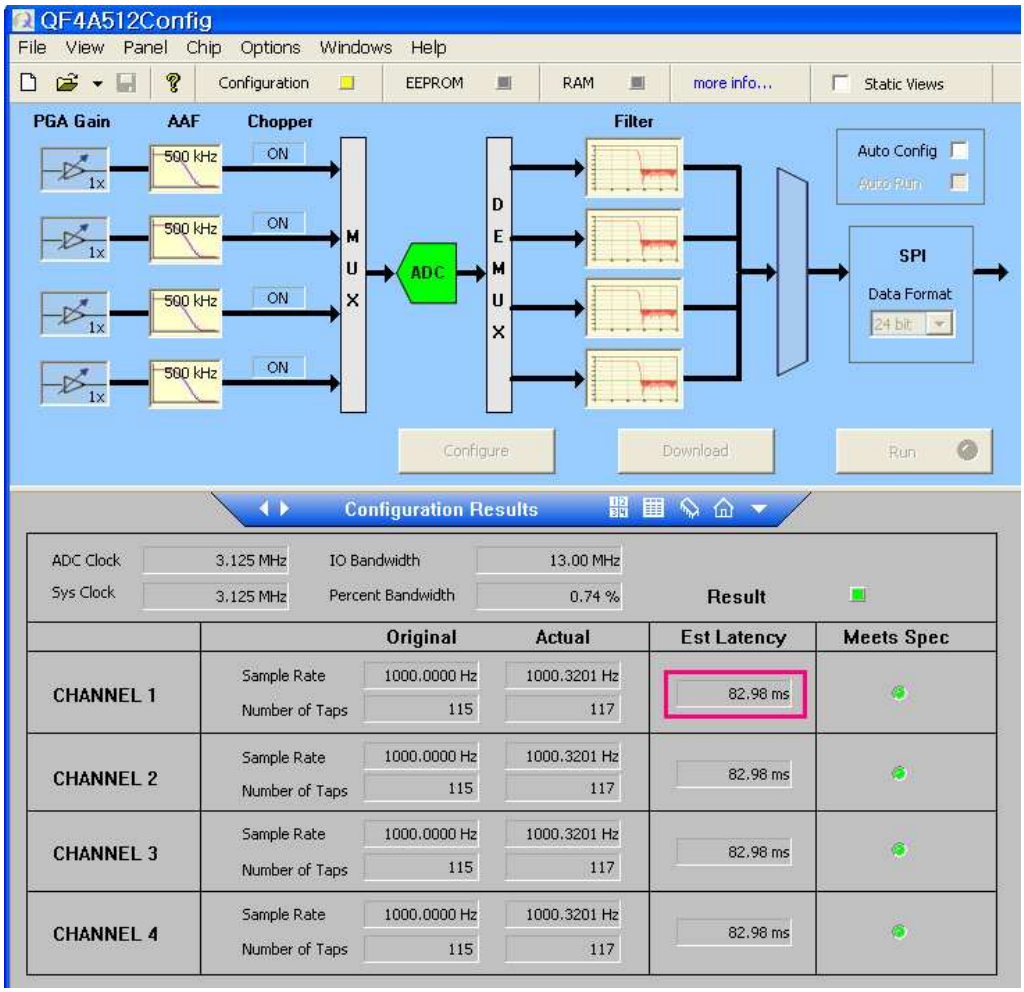

Figure 10. Latency

2. Calculate the equivalent number of points:
   pts = latency * sample rate
   eg: 82.98 ms * 1000 Hz = 83 (rounded integer)

3. Change delayconstant in `filters.h` (shm/sensorboards/SHM_A/filter.h) to the calculated value as shown in Figure 11.

```
90 #ifndef _FILTERS_H
91 #define _FILTERS_H
92
93 #include "filterCustom.h" /* must be included first - defines QFRegister */
94 #include "filter3ch_fs10Hz_fc4Hz.h"
95 #include "filter3ch_fs50Hz_fc20Hz.h"
96 #include "filter3ch_fs100Hz_fc40Hz.h"
97 #include "filter3ch_fs280Hz_fc70Hz.h"
98 #include "calibrationFilter.h"
99
100 #ifndef SHMA_CUSTOM_FILTER_RATE
101 #define SHMA_CUSTOM_FILTER_RATE (1000)
102 #endif
103
104 const int *QFImageRegisterTableSizeArray[] = {
105                                                 &QFImageRegisterTableSize10,
106                                                 &QFImageRegisterTableSize50,
107                                                 &QFImageRegisterTableSize100,
108                                                 &QFImageRegisterTableSize280,
109                                                 &QFImageRegisterTableSize };
110
111 const struct QFRegister *QFImageRegisterTableArray[] = {
112                                                 QFImageRegisterTable10,
113                                                 QFImageRegisterTable50,
114                                                 QFImageRegisterTable100,
115                                                 QFImageRegisterTable280,
116                                                 QFImageRegisterTable };
117
118 const uint32_t QFDelayArray[] = {
119                                   80,//for 10Hz config
120                                   89,//for 50Hz config
121                                   78,//for 100Hz config
122                                   94,//for 280Hz config
123                                   83 };//for custom filter config
124
125 #endif
```

Figure 11. Registration of delay constant for user-defined filter (e.g. 83 for 1000hz)

3 Temperature Calibration and Correction

A capacitive sensor is susceptible to mean value drift of the signal due to the temperature change inside the sensor. The LIS344ALH accelerometer that is used on the ISM400 board also experiences this drift effect, according to the datasheet (http://www.st.com/stonline/products/literature/ds/14337.pdf), the zero-g level change rate for temperature change of LIS344AH is ±0.5mg/°C. To address this phenomenon, onboard temperature compensation has been implemented in the software.

The availability of the onboard temperature sensor allows the temperature to be measured simultaneously with the acceleration. The result is that the direct relationship between the self heating of the board and the accelerometer can be determined. The accelerometer and the temperature sensor are located similar distances from the most heat-generating components, the QF4A512 and the Imote2 Processor as indicated in Figure 12, so the temperature sensor is expected to read temperatures very similar to those experienced by the accelerometer.

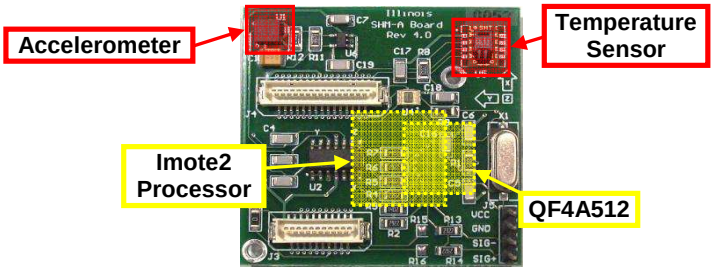

Figure 12. Location of components on the ISM400 sensor board.

Figure 13 shows an example of the temperature measured on board the ISM400 sensor board over a two minute period. The correlation between the temperature and the zero-g drift (mean value) can be determined by plotting the mean value versus temperature and using linear regression to fit a line to the data as shown in Figure 13. The slope of the line is the zero-g offset drift as a function of temperature. In the example data shown in the figure below, the *x*-axis mean value temperature sensitivity is 0.555 mg/°C and the *y*-axis mean value temperature sensitivity is 0.431 mg/°C.

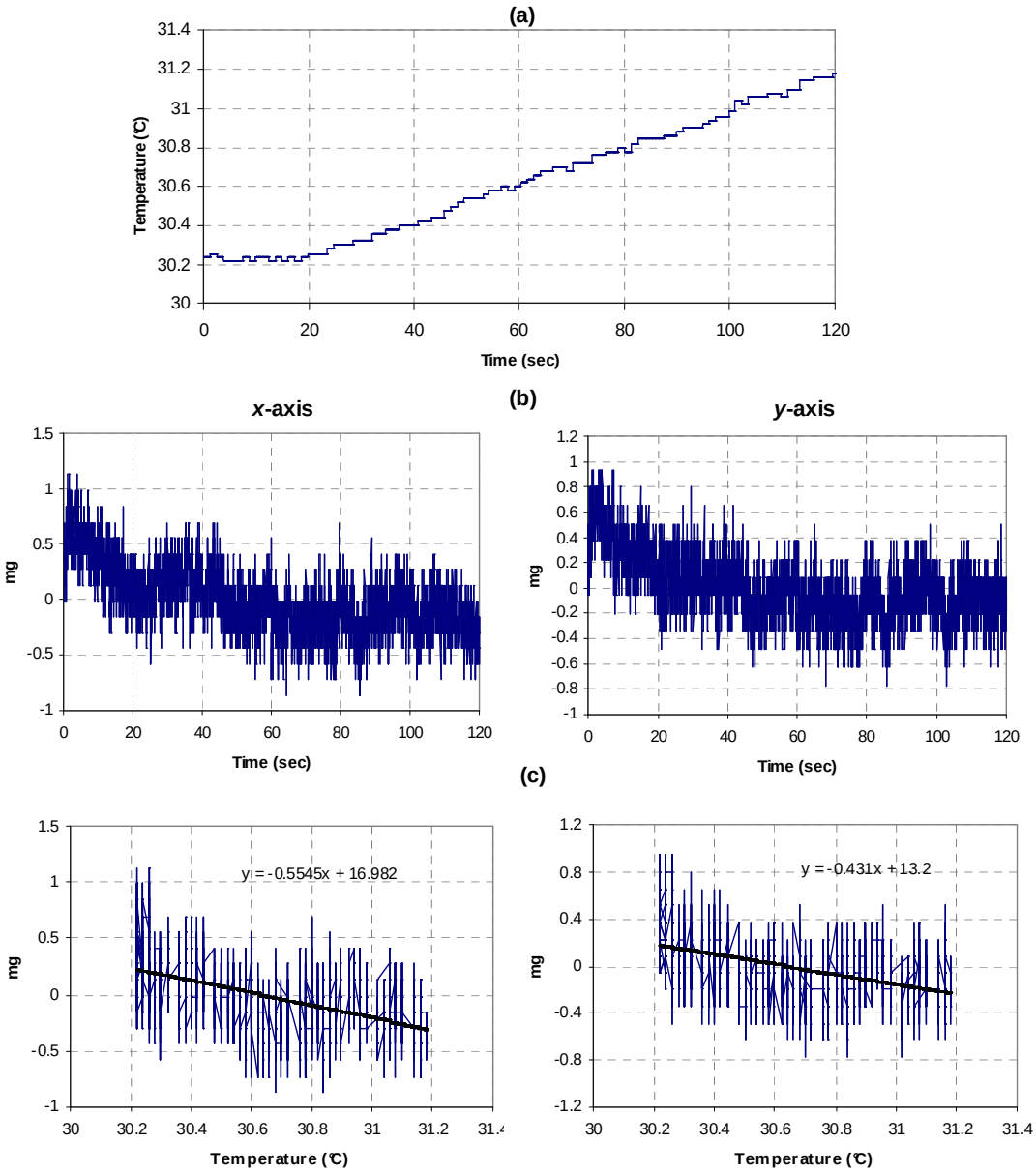

Figure 13. Measured temperature on ISM400 sensor board (a), uncorrected accelerometer readings (b) and acceleration vs. temperature plots (c).

3.1 Determine mean value temperature sensitivities

The software driver for the ISM400 sensor board has some default mean value temperature sensitivity values built in based on testing of over 100 boards. There is, however, variation in the mean value temperature sensitivity from board to board, resulting in potential error if the default values are retained. If more accurate temperature compensation is required, the mean value sensitivities of each channel on each sensor board can be determined and implemented using `SHMATempCal` utility. The `SHMATempCal` utility can be run on up to 40 remote nodes at

one time, facilitating efficient sensitivity constant determination for each channel on many sensor boards.

- Before issuing any commands, make sure your application includes `SHMATempCal` utility as shown in Figure 14.

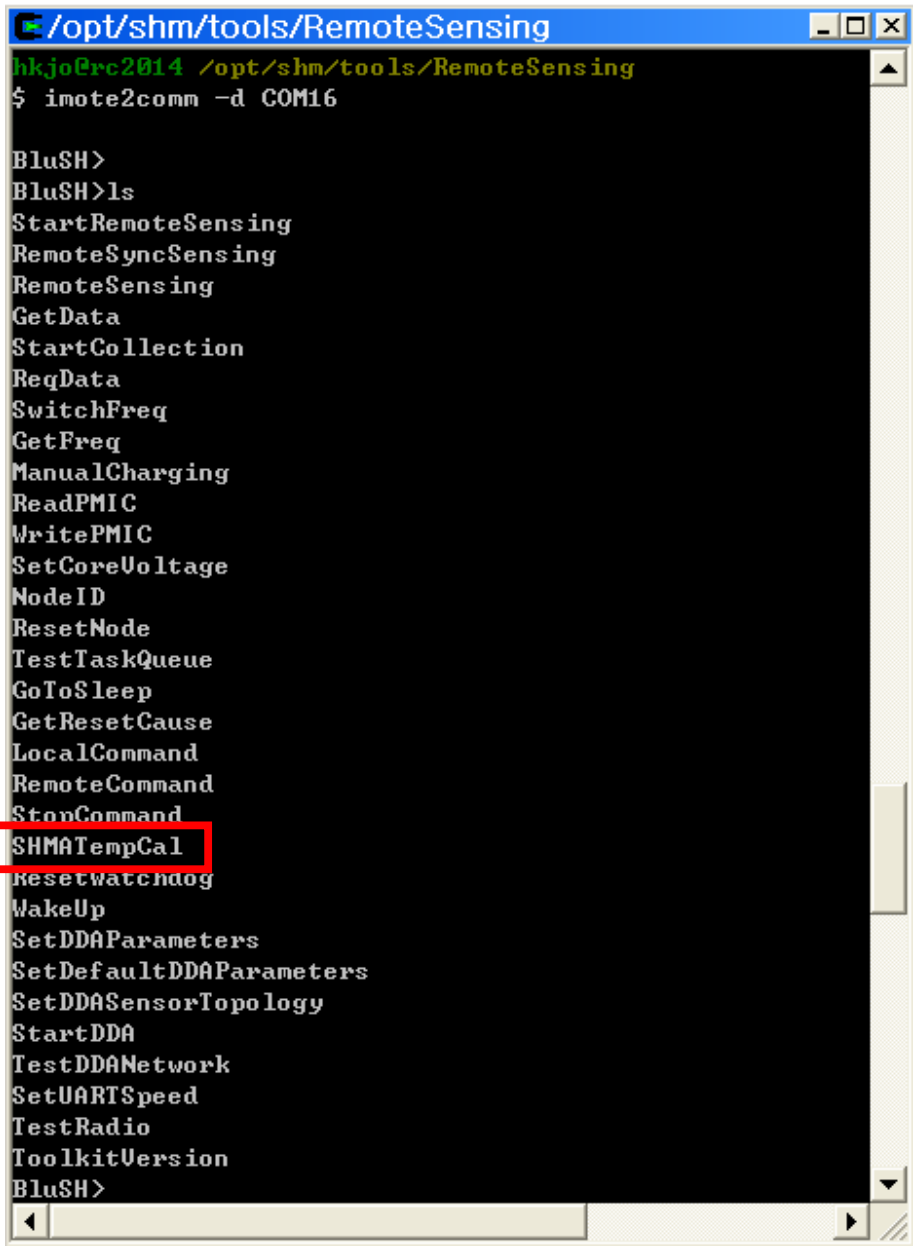

Figure 14. Checking available Blush commands.

- To get the temperature sensitivity constants, type the following command at the `BluSH>` prompt:
  - `SHMTempCal <nodeId> [nodeId] [nodeId] …`

- Then, the temperature sensitivity constants are given as shown in Figure 15 for each channel of each leaf node,

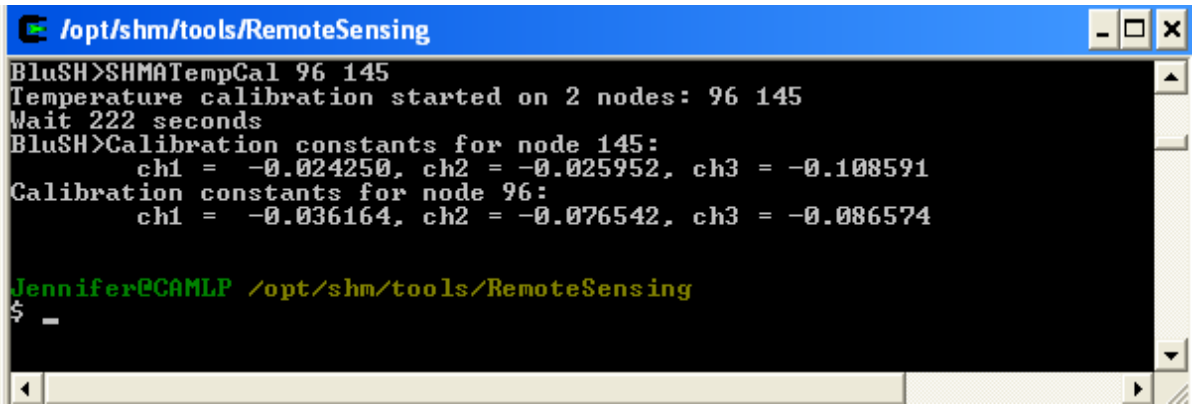

Figure 15. Example outputs from the SHMTempCal utility

- **Note**: If the calibration utility is run more than once, the results may vary slightly from test to test due to inherent signal noise and temperature measurement error (up to ±0.4°C according to the SHT11 datasheet, http://www.sensirion.com/en/pdf/product_information/Datasheet-humidity-sensor-SHT1x.pdf) so the sensitivity values returned by the utility are approximate. The values produced by the utility do not have the units of mg/°C but rather are scaled to convert the raw ADC values.

3.2 Update the ISM400 driver files

To implement the individually recorded temperature sensitivities for acceleration sensing, the values resulting from SHMATempCal should be registered in the driver file.

- Open the sensorboard.h file (shm/sensorboards/SHM_A/sensorboard.h), and change the temperature sensitivity constants as shown in Figure 16.
- Each time these constants are changed in sensorboard.h, the sampling application (eg. RemoteSensing) should be recompiled and reinstalled on each corresponding leaf node.
- Figure 17 shows that the mean value of the ISM400 board output does not drift in time after temperature compensation.

```
const sensor_rates_t accelChannelRates = {5, {10., 50., 100., 280., SHMA_CUSTOM_FILTER_RATE}};
const sensor_scaling_t accelChannelScaling = {0.144, 0.0};      // need to check for all 3 axis
const sensor_scaling_t tempChannelScaling = {0.01, -39.60};
const sensor_rates_t tempChannelRates = {1, {0.}};

const sensor_rates_t *supportedSamplingRates[TOTAL_SENSOR_CHANNELS] = {
    &accelChannelRates,
    &accelChannelRates,
    &accelChannelRates,
    &accelChannelRates,
    &tempChannelRates,
    &tempChannelRates
};

const sensor_scaling_t *channelDataScaling[TOTAL_SENSOR_CHANNELS] = {
    &accelChannelScaling,
    &accelChannelScaling,
    &accelChannelScaling,
    &accelChannelScaling,
    &tempChannelScaling,
    &accelChannelScaling
};

const float tempSens[] = {0.024, 0.026, 0.109};

#endif /* _SENSORBOARD_H */
```

Figure 16. Registration of temperature sensitivity constants in `sensorboard.h`

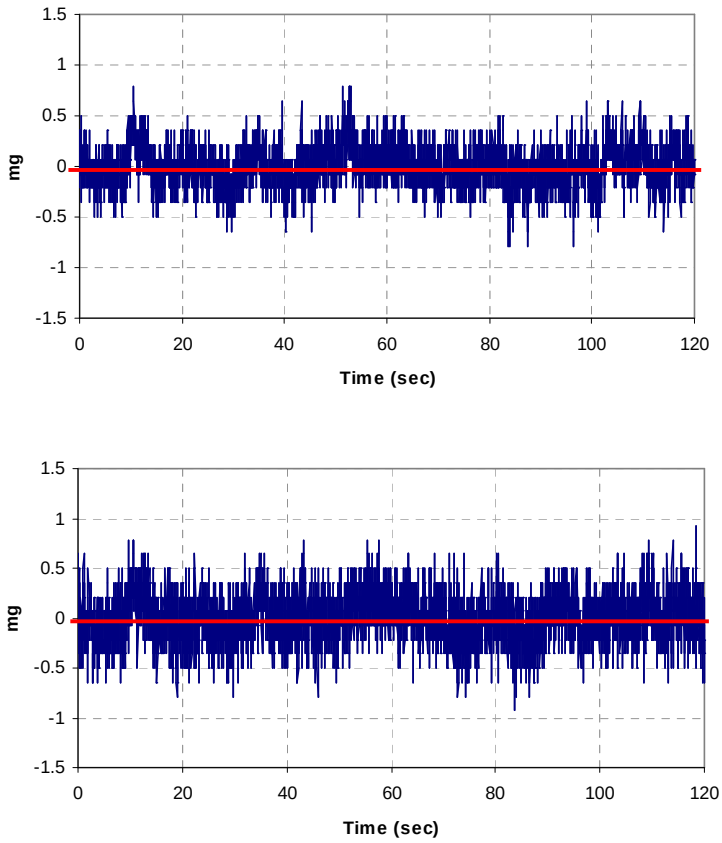

Figure 17. Scaled mean value measurements of *x*-axis (top) and *y*-axis (bottom) collected after onboard temperature correction.

For inquiries, please contact:

Professor B.F. Spencer, Jr.
bfs@illinois.edu
University of Illinois at Urbana-Champaign
Department of Civil and Environmental Engineering
2213 Newmark Civil Engineering Laboratory, MC-250
205 North Mathews Ave
Urbana, IL 61801
USA

Or visit:

http://shm.cs.uiuc.edu