

Wireless Structural Health Monitoring System Design, Implementation and Validation

Harshvardan P. Joshi*, Bogdan Burlacu†, Manjunath M. Prabhu*, Suyoung Yoon*,
Mihail L. Sichertiu*, Rudra Dutta‡ and Sami H. Rizkalla†

NC State University,
Raleigh, NC 27511

December 23, 2005

Abstract

This document describes the development of a structural health monitoring system using wireless sensor nodes equipped with strain gauges and its validation. Each node consists of a sensor (strain gauge), a signal conditioning circuit and a mote¹. Several such nodes can be monitored and controlled by a base station. The system measures strain in the structural element, converts it to an analog voltage signal, amplifies and filters the analog signal, converts it to a digital signal, processes and, finally, transmits it using wireless links. Each of the building blocks of the system, its functions, design considerations and specifications are described in this report.

1 Introduction

Measurements of surface deformation are commonly used to monitor the health of any structure. Resistive strain gauges are the most popular way of measuring such deformation or strain [1]. Often, it is useful to monitor the health of remote structures from a central monitoring and control station at a low cost. With recent advances in wireless sensor networking this is now feasible. Wireless sensor networks offer several advantages over traditional wired setups:

- *Low Cost*: motes are relatively cheap.
- *Wireless Connectivity*: eliminates the need for laying costly and disconnection-prone cables from remote sites to the monitoring station.
- *Fast Deployment, Flexible Topology*: since the sensor network does not require any fixed infrastructure and forms its own network (an ad-hoc network), it can be deployed very fast. Similarly the number and location of the monitoring sites can be dynamically changed without any efforts to reconfigure the network.
- *Minor Development Efforts*: the only significant development effort required is a signal conditioning circuit and software that interfaces the strain gauge sensor with the mote.
- *Low Maintenance & Operating Cost*: since sensor nodes consume very little power, are robust, and can be reprogrammed and calibrated from a remote location, they require very little on-site maintenance.

*Department of Electrical and Computer Engineering

†Department of Civil, Construction, and Environmental Engineering

‡Department of Computer Science

¹See *Glossary* for definition.

This report describes the development of a structural health monitoring system using wireless sensor nodes with strain gauges. The system consists of several nodes connected through (multi-hop) wireless links to a base station used for monitoring. Each node has a sensor (strain gauge), a signal conditioning circuit and a mote. A logical block diagram of the system is shown in Fig. 1. The following sections describe each major block.

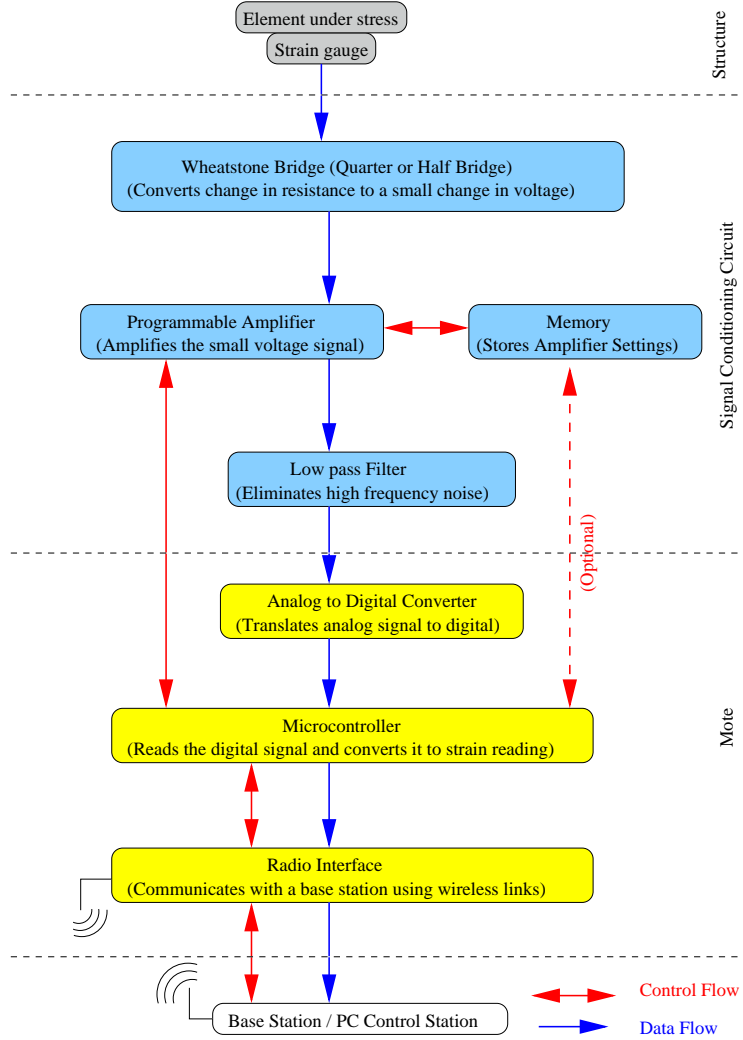


Figure 1: Block diagram of wireless structural health monitoring system

2 Strain Gauge

A strain gauge is a sensor designed specifically to measure mechanical strain. The most popular strain gauge is a resistive element bonded to the element where stress is applied. The resistance of the gauge changes with the strain in the element. The relation between the change in resistance and strain is given by:

$$\epsilon_a = \frac{\Delta L}{L} = \frac{\Delta R/R}{G_f}, \quad (1)$$

or

$$\frac{\Delta R}{R} = \epsilon_a G_f, \quad (2)$$

where ϵ_a is the axial strain, L is the nominal length of the strain gauge, R is the nominal resistance of the strain gauge and G_f is the gauge factor. ΔL and ΔR are the change in the length and resistance of the strain gauge due to applied strain respectively. The *gauge factor* G_f describes a gauge's sensitivity to strain. A detailed treatment on stress-strain relationship and strain gauges can be found in [1].

The following characteristics of strain gauges can significantly affect the performance of the proposed system:

- The *dimensions* of the strain gauge determine the area over which the strain is actually measured. If the measured strain is averaged over a larger area it is more likely to be representative. However, the dimensions of the strain gauge also affect its sensitivity to transverse strain.
- The *gauge factor*, G_f , determines the sensitivity of the gauge to strain. Most of the commercially available strain gauges have a G_f of around 2 [1].
- The effect of *temperature* changes can be significant on the strain gauge. The nominal resistance of the gauge varies with the temperature, and this results in what is called *apparent strain*. The apparent strain is also produced by the difference in thermal expansion coefficients of the gauge and the base material.
- The *linear* relationship between strain and change in resistance may not hold over an extended range. For this project the range of strain to be measured is $\pm 2000 \mu\text{strain}$ corresponding to a deflection of $\pm 0.2\%$ from the nominal value. Hence, we can assume the strain gauge to be linear for the considered range.

2.1 Design

The strain gauge used for this project has a nominal resistance of 1000Ω with a gauge factor G_f of 2.105.

3 Signal Conditioning Circuit

The change in resistance induced by strain in the strain gauge has to be converted to voltage such that the information can be processed by a computer. The signal conditioning circuit converts the change in resistance to change in voltage, amplifies the signal, adjusts the offset, linearizes the signal, filters out noise, and makes the signal suitable to be read by the analog to digital converter (ADC) on the mote. In the following sections we describe how each of these functions is designed and implemented.

3.1 Wheatstone Bridge

A Wheatstone bridge is employed to measure the small resistance change in a strain gauge. Since the change in resistance is very small (less than $\pm 1\%$) the bridge output can be considered to be linear even when the bridge is not balanced. The Wheatstone bridge can be either a *quarter bridge* or a *half bridge*. The quarter bridge has only one active arm, i.e., strain gauge, and it is the most simple form of bridge. The half bridge has two active arms, and it has inherent temperature compensation capability and higher sensitivity than the quarter bridge.

3.1.1 Design

We are using a quarter bridge for our project, though a half bridge can also be used². A quarter bridge is shown in Fig. 2. The arm marked R_g is the strain gauge. The nominal output of the bridge is given by:

$$V_{diff} = \left(\frac{R_g}{R_1 + R_g} - \frac{R_4}{R_3 + R_4} \right) V_{EXC}. \quad (3)$$

²Our circuit board allows the use of a half bridge by selecting certain jumpers. For details refer to section 3.6

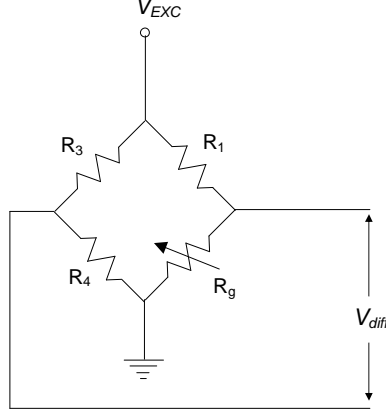


Figure 2: Wheatstone bridge in quarter bridge configuration.

In the absence of strain, since the value of R_1 is same as R_3 and the value of R_g is same as R_4 , the bridge output is zero. However, when strain is applied to the strain gauge, its resistance changes by a small amount, ΔR . The output of the bridge is:

$$V_{diff} = \left(\frac{R_g + \Delta R}{R_1 + R_g + \Delta R} - \frac{R_4}{R_3 + R_4} \right) V_{EXC} \quad (4)$$

In (4), ΔR appears both in the denominator as well as the numerator. Thus, the bridge output is not linear with respect to ΔR . However, since the term ΔR is very small (less than $\pm 1\%$), we can neglect this non-linearity. If needed, the non-linearity can be compensated by the microcontroller.

The output of the bridge, V_{diff} , is directly proportional to the excitation voltage V_{EXC} . Hence, output can be increased by increasing V_{EXC} . Similarly, by reducing the values of R_1 and R_3 with respect to the nominal value of R_g , the sensitivity of the bridge can be increased. However, lower values for R_1 and R_3 will result in increased power consumption. A bridge excitation of 2.075 V is provided by an internal voltage regulator of the programmable amplifier (see Section 3.2 for details).

3.1.2 Illustration

For our project the nominal values are: $R_1 = R_3 = 5000\Omega$; $R_g = R_4 = 1000\Omega$; $V_{EXC} = 2.075V$

Let us calculate the maximum output of the bridge, i.e., the bridge output at full scale. Since the maximum strain applied, $\epsilon_{a_{max}}$, is 2000 μ strain and the gauge factor G_f of the strain gauge used is 2.105, using (2) the maximum change in strain resistance is $\Delta R_{max} = 4.21\Omega$

From (4), and with these values, we obtain $V_{diff_{max}} = 1.212448mV$

Similarly we can calculate the minimum V_{diff} we must be able to measure if we need a resolution of 1 μ strain. In that case, $\Delta R_{min} = 2.105 \times 10^{-3}\Omega$ and $V_{diff_{min}} = 0.606649\mu V$

The Full Scale Sensitivity (FSS) of the bridge is specified in V/V or mV/V , and gives a measure of bridge output per unit volt of excitation. It is given by,

$$FSS = \frac{V_{diff_{max}}}{V_{EXC}} = \left(\frac{R_g + \Delta R_{max}}{R_1 + R_g + \Delta R_{max}} - \frac{R_4}{R_3 + R_4} \right) V/V \quad (5)$$

With the values in our setup we obtain, $FSS = 0.584312mV/V$.

The power consumption of the bridge at no load is given by,

$$P_{bridge} = I_{bridge}^2 R_{bridge} \quad (6)$$

where, I_{bridge} is the total current drawn by the bridge and R_{bridge} is the total effective resistance of the bridge. The effective bridge resistance R_{bridge} in the absence of strain is 3000Ω , and the total current I_{bridge} is 0.6917mA at V_{EXC} of 2.075V. Hence the total power consumption at no load is, $P_{bridge} = 1.435mW$. The power consumption at full load is not significantly different, as the change in resistance is very small.

3.2 Programmable Amplifier

As shown in the previous section, the maximum output of the Wheatstone bridge is very small, and it cannot be increased by changes in the bridge. Hence, it is necessary to amplify the output signal of the bridge so that it can be further processed. We use a programmable amplifier (PGA) rather than a simple OP-AMP, as it has the following advantages:

- *Programmable Gain*: the gain of the amplifier can be changed dynamically, without any change in the hardware. This is useful in changing the dynamic range of the system. For example, we may choose to double the resolution (and have a range of only $\pm 1000\mu\text{strain}$) by simply doubling the gain.
- *Programmable Offset*: the offset of the amplifier can be changed dynamically, without any change in the hardware. This feature is very useful for our project as the strain measurement range is $\pm 2000\mu\text{strain}$, which means that bridge may produce both positive as well as negative output while the ADC on board the mote can accept only positive signals. By setting the offset in the middle of the range, we can divide the ADC input range to a virtual negative part.
- *Ease of Calibration*: due to the adjustable gain and offset, the calibration of the transducer becomes very easy and can be done from a remote location.
- *Temperature Compensation*: the programmable amplifier has an in-built temperature sensor which can be used for temperature compensation (Alternatively, external temperature sensor can be used for compensation.). The temperature compensation can be done by providing a table of gain and offset to be used for different temperatures.
- *Bridge Excitation*: the programmable amplifier has an in-built voltage regulator which can be used as a constant voltage source for the Wheatstone bridge.
- *Over-Scale and Under-Scale Limits*: the programmable amplifier has in-built over/under-scale protection circuit which can be used to limit the output of the amplifier within some percentage of the ADC range. This not only helps in protection of ADC but also helps in over-scale or under-scale indication or fault diagnosis.

3.2.1 Design

We use PGA309, a programmable amplifier from Texas Instruments, in our signal conditioning circuit. Here we discuss the most relevant design considerations in programming PGA309. More details about PGA309 can be found in the datasheet [2] and users guide [3].

PGA309 has basically two amplifiers: a front-end amplifier with gain selectable from 4 to 128, and an output amplifier with selectable gain of 2 to 9. There is also a 16-bit gain DAC, which is used for fine gain adjust. The maximum gain provided by PGA309 can be 1152. The offset in PGA309 can be adjusted at two places: the coarse offset is adjusted before the front-end amplifier while the fine offset adjust is just after the front-end amplifier but before the fine gain adjust.

PGA309 needs a stable reference voltage and can use either an external reference or an internal voltage reference. We use the internal reference (selectable as 2.5V or 4.096V) at 2.5V. The bridge excitation is provided through the V_{EXC} pin. PGA 309 is configured to provide a constant supply of 2.075 V to the Wheatstone bridge. It is possible to vary V_{EXC} to linearize the output of the sensor, although this feature is not used in this project.

PGA309 has two types of digital interfaces. One is a single pin (PRG) UART compatible interface with bit rate from 4.8 kbps to 38.4 kbps. Another is a two pin (SDA & SCL) two wire standard interface with the clock speed of 1 kHz to 400 kHz. The PRG pin can be tied to V_{OUT} pin to operate in a true 3-wire sensor³ mode. However, this option is not used here because the power is provided locally by the mote and communication with remote location is through a wireless link.

³Normally, four wires run between a remote sensor and the control station – two for power and two for analog signal. However, in non-hazardous environments, power and signal can have a common ground and one wire can be eliminated. Sensor in this configuration is called a 3-wire sensor. To program the PGA309 through PRG pin an extra wire may be needed. However, by tying PRG pin with V_{OUT} and alternately enabling one of these pins, this extra wire can also be eliminated.

PGA309 has nine internal registers that are used to configure the amplifier as well as read its status. It is also interfaced with an external EEPROM where some of the configuration data is stored and where the temperature compensation look up table is also stored. On power-up, the PGA resets all its internal registers and reads the configuration data from the EEPROM.

Register Description

1. *Register 0*: Temperature ADC Output, 12-bit in internal temperature mode and 15-bit in external temperature mode. We use the internal temperature sensor, which gives a resolution of $0.0625^{\circ}\text{C}/\text{count}$. Temperature can be read from this register for temperature compensation. This is a Read Only register.
2. *Register 1*: Fine Offset Adjust (Zero DAC) Register. When set to 0x0000 it produces $0 V_{REF}$ output, and 0xFFFF produces $0.9999847 V_{REF}$ output. The values will be set during calibration, and can also be changed dynamically as necessary.
3. *Register 2*: Fine Gain Adjust (Gain DAC) Register. When set to 0x0000 it produces 0.333333333 gain, while 0xFFFF produces 1.000000000 gain. The values will be set during calibration, and can also be changed dynamically as necessary.
4. *Register 3*: Reference Control and Linearization Register. Used to control the reference and excitation voltages and for linearization settings. The following bits are relevant:
 - **EXS**: Reset to 0 to select Range 1, which will produce V_{EXC} of 2.075V.
 - **EXEN**: Set to 1 to enable V_{EXC} .
 - **RS**: Set to 1 to select internal V_{REF} of 2.5V.
 - **REN**: Set to 1 to enable internal voltage reference.
 - **LD[7:0]**: Reset to all zeros. This is for linearization DAC settings. We are not using sensor linearization DAC, as the strain gauge is linear for the given range.
5. *Register 4*: PGA Coarse Offset Adjust and Gain Select Register. Used to set the coarse offset and to select the gain of front-end and output amplifiers. The following bits are relevant:
 - **OWD**: Set to 1 to enable one-wire digital interface through PRG pin.
 - **GO[2:0]**: Output amplifier gain select. 000_2 produces a gain of 2 while 110_2 produces a gain of 9.
 - **GI[3:0]**: Front-end amplifier gain select. GI[3] is for gain polarity, while GI[2:0] is used to select a gain from 4 to 128.
 - **OS[4:0]**: Coarse offset adjust on front-end amplifier. OS[4] is for polarity, while OS[3:0] are used to select an offset as a function of V_{REF} .
6. *Register 5*: PGA Configuration and Over/Under-Scale Limit Register. Used to configure miscellaneous properties of PGA and for over/under-scale limit settings. The following bits are relevant:
 - **CLK_CFG[1:0]**: Reset to 00 (Typical clock).
 - **EXTEN**: Set to 1 to enable external fault comparator group.
 - **INTEN**: Reset to 0 to disable internal fault comparator group.
 - **EXTPOL**: Set to 1 to force V_{OUT} high when external fault comparator detects a fault.
 - **INTPOL**: Reset to 0 to force V_{OUT} low when internal fault comparator detects a fault.
 - **OUEN**: Set to 1 to enable over/under-scale limits.
 - **HL[2:0]**: Reset to all zeros to select over-scale limit of $0.9708V_{REF}$.
 - **LL[2:0]**: Reset to all zeros to select under-scale limit of $0.02540V_{REF}$.
7. *Register 6*: Temperature ADC Control Register. Used to control the Temp ADC. The following bits are relevant:

- **ADC2X**: Reset to 0 for 1x conversion speed.
- **ADCS**: Set to 1. Doesn't matter when **CEN** is set.
- **ISEN**: Reset to 0 to disable internal current source.
- **CEN**: Set to 1 for continuous conversion mode.
- **TEN**: Set to 1 to enable internal temperature mode.
- **Other bits**: Other bits are set as: **AREN**: 0, **RV[1:0]**: 00, **M[1:0]**: 00, **G[1:0]**: 00, and **R[1:0]**: 11. For details refer to the user's guide [3].

8. *Register 7*: Output Enable Counter Control Register. This register is used to set the output enable counter to enable or disable V_{OUT} when the PRG pin is connected to it in a true three-wire sensor mode. This is not used in our project.

9. *Register 8*: Alarm Status Register. It is a Read Only register, where the fault monitor comparator outputs are written by the PGA.

3.2.2 Power Consumption

The PGA309 draws a quiescent current of 1.2mA to 1.6mA at 5V without the bridge load. Thus the quiescent power consumption of the chip is 8mW.

3.3 Memory (EEPROM)

The EEPROM is used to store the configuration for the registers of the PGA, and, optionally, to store a temperature coefficient lookup table for temperature compensation. On power-up, the PGA resets all its internal registers and reads the EEPROM for the configuration data. Hence, it is necessary to have the EEPROM and ensure that proper configuration data is written in it. The PGA also reads the relevant fine gain and fine offset adjust parameters for current temperature from the EEPROM.

3.3.1 Design

We use 24LC16B, a 16 Kbit EEPROM from Microchip Technologies, for our circuit. 24LC16B can operate at low voltages and has a low active (1mA) and standby ($1\mu\text{A}$) current, thus suitable for battery powered applications. It has a two-wire serial interface which is used to communicate with the PGA, and can also be used to interface with the MCU on the mote. It is designed for 1,000,000 erase/write cycles, and is available in DIP as well as surface mount packages. More details can be found in the datasheet [4].

3.4 Filter

The output of the programmable amplifier is fed to a low-pass filter to eliminate high frequency noise. The main sources of noise are:

- *Vibration*: We are interested in measuring strain which is a relatively slowly varying quantity. However, the structural element is also subjected to high frequency vibration.
- *AC Power Lines*: Nearby AC power lines may induce 60 Hz noise in the components and wires of the circuit.
- *RF noise*: The circuit may receive any RF signal present in the environment.

3.4.1 Design

We use a 2^{nd} order Butterworth filter with a cut-off frequency of 40 Hz as the low-pass filter in Sallen-Key topology [5]. We use AD822, a quad Op Amp package, as the Op Amp for the filter. The specifications of AD822 can be found in the data sheet [6]. The filter has been designed using the *Analog Filter Wizard* tool [5] provided by Analog Devices. The values of the passive components of the filter are shown in the schematic of Fig. 3. AD822 draws 1.6mA at 5V supply voltage, i.e. power consumption of 8mW.

3.5 CMOS Inverter

A CMOS inverter is basically a NOT gate, with CMOS logic voltage levels. The CMOS Inverter serves two functions:

- interface the single pin (PRG) of the PGA with the two pin UART of the mote;
- control the power of signal conditioning circuit so that software controlled power saving can be done .

Since the PGA has only a single pin for communication with UART, both receive(Rx) and transmit(Tx) have to be transmitted through this pin. If the Rx and Tx pins of mote UART are connected to this pin without any interfacing circuit, then if the PGA tries to transmit, the Tx pin of the UART drains the power and doesn't allow the bus to be pulled high. Thus we have to isolate the Rx and Tx pins of mote, while using the PRG pin at the same time. Putting a pair of CMOS inverters on both Rx and Tx pins of UART before connecting to PRG pin, as shown in Fig. 3, solves the problem. Even when the PGA is transmitting and pulls up the PRG pin, the Tx pin of the mote remains isolated because of the CMOS gates and hence can float freely without draining power from the PGA.

Since the sensor nodes are expected to remain unsupervised for long periods of time after deployment, they have to run on very limited power. Motes typically conserve energy by going to sleep for a long period of time. In the signal conditioning circuit, even when a measurement is not being done (i.e., even when ADC is not reading the output), the bridge continuously drains power. Hence, it is important to provide the capability of putting the entire conditioning circuit to sleep through software. To achieve this, power to the whole circuit is supplied through one of the CMOS inverters, controlled by one of the I/O pins of the mote. Thus by toggling the I/O pin (through software), we can turn the power to signal conditioning circuit ON or OFF.

We use SN74HC04, a hex inverter (i.e., six inverters in a single chip) with low power consumption and capacity to drive $\pm 4mA$ load at $5V$ on each of the inverter. The power consumption of the signal conditioning circuit is $3.0mA$ to $3.5mA$ at $3V$, which can be supplied by an inverter. It has a maximum I_{CC} of $20\mu A$ at $6V$, and typical switching time of $8ns$ at $6V$.

3.6 Circuit Board

The schematic of the signal conditioning is presented in Fig. 3. A list of parts is available in the Appendix.

The circuit board on which the signal conditioning circuit been implemented (shown in Fig. 5) is a 4-layer board, 5 cm x 3.2 cm (approximately. 2 in x 1.2 in) in size. It connects to the mote by a standard 10-pin header. The layout of the circuit board is shown in Fig 4(a). In addition to the basic signal conditioning circuit, it has four jumpers which can be used to configure the circuit and operate it in specific modes. The jumpers seen in Fig 4(b) are described below. Please note that in Fig 4(b), each jumper (named JP1 through JP4) has four pins and there is a rectangular box between two of its pins. These two pins are the pins numbered 1 & 2 in the schematic of Fig. 3, while the other two pins are pins 3 & 4.

- *Jumper 1*: Labeled JP1 on the circuit board, it can be used to select whether to operate the PGA in a *Test* mode or in normal mode. When JP1 connects pins 1 & 2, it pulls the TEST pin on PGA309 high, putting it in the *Test* mode. In the *Test* mode, PGA does not periodically read EEPROM for fine gain and offset values. Thus, PGA retains the programmed gain and offset. This mode is useful when calibrating the circuit or testing the system. When JP1 connects pins 3 & 4, it pulls the TEST pin on PGA309 to ground, thus making it operate in the normal mode. (Note: For proper operation of the circuit, with the current software configuration, JP1 should connect pins 1 & 2)
- *Jumper 2*: Labeled JP2 on the circuit board, it can be used to select the digital interface to be used to program the PGA. The PGA has two digital interfaces as described in section 3.2.1. PRG, the single pin UART compatible interface is used to program the mote using UART0 of the mote. However, in future the I^2C interface of the PGA (SDA & SCL pins) can also be used by connecting it to the I^2C bus of the mote through JP2. To use the I^2C interface, JP2 should connect pins 1 & 2, as well as pins 3 & 4.

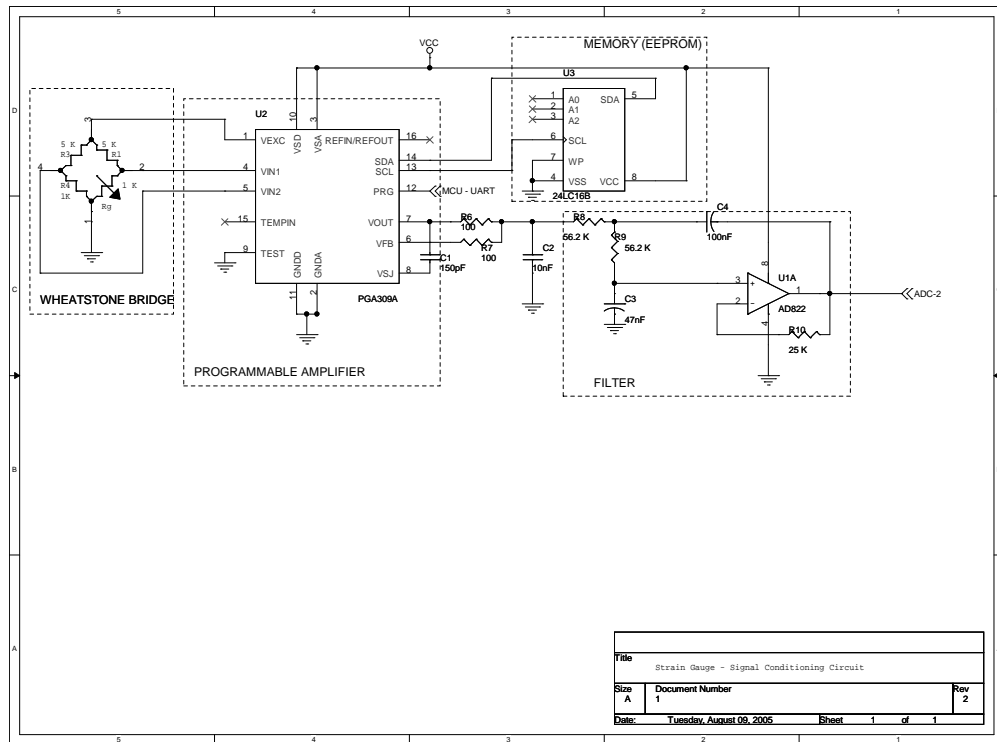


Figure 3: Schematic diagram of the signal conditioning circuit.

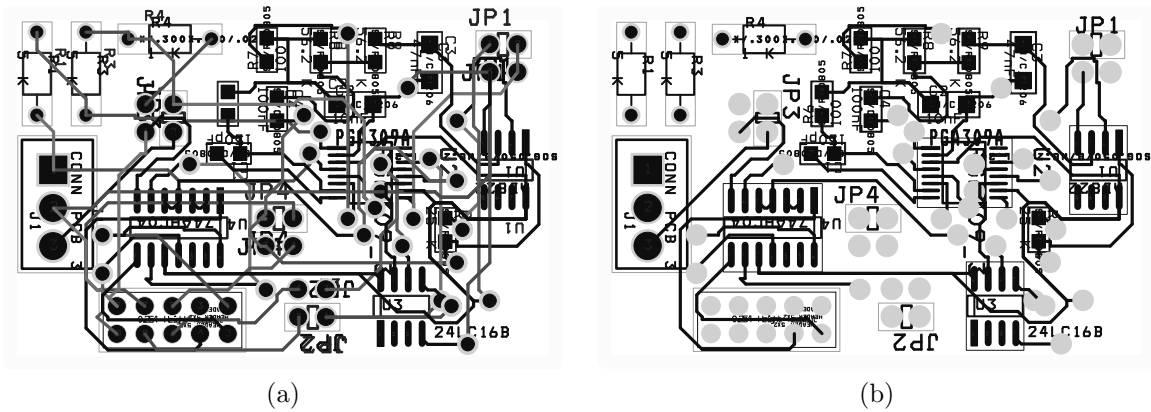


Figure 4: The PCB layout of Signal Conditioning Circuit (a) all layers and (b) top layer

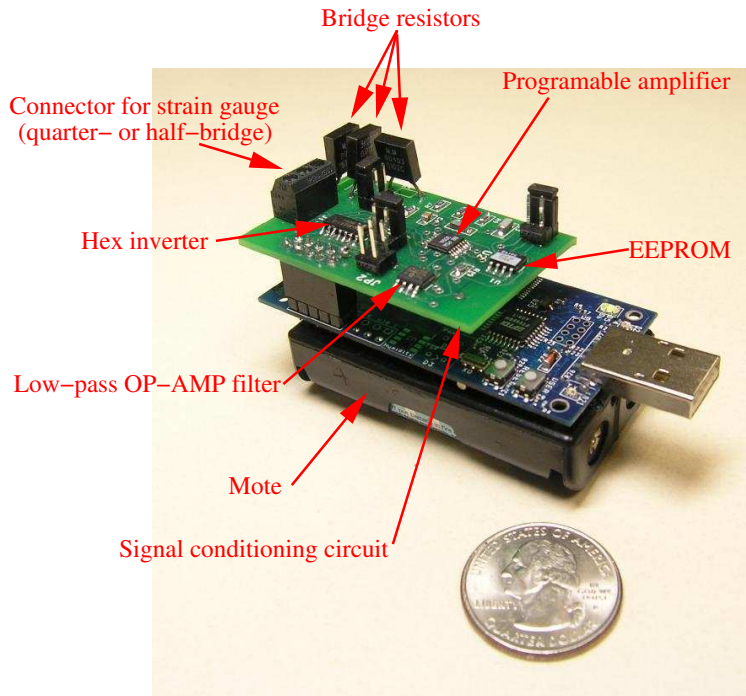


Figure 5: The circuit board of the signal conditioning circuit connected to a Tmote Sky.

- *Jumper 3*: Labeled JP3 on the circuit board, it can be used to select either quarter-bridge or half-bridge mode of operation. When JP3 connects pins 3 & 4, the bridge uses on-board resistor R_4 and has only one active arm, thus operating as a quarter-bridge. If JP3 connects pins 1 & 2, the resistor R_4 is bypassed, and two active arms can be connected to bridge, which increases the bridge sensitivity and inherently compensates for temperature variations. (Note: In quarter-bridge mode, the strain gauge should be connected to pins 1 & 2 of the terminal block (labeled CONN PCB 3 on the layout); in half-bridge mode, the two strain gauges should be connected to pins 1 & 2 and 2 & 3.)
- *Jumper 4*: Labeled JP4 on the circuit board, it can be used to select the ADC on board the mote to which the output signal of the signal conditioning circuit is fed. When JP4 connects pins 1 & 2, the output signal goes to ADC2 of the mote, while when JP4 connects pins 3 & 4, the signal goes to ADC1. (Note: The mote software is configured to read ADC2, hence JP4 should connect pins 1 & 2)

4 Mote

The Mote is the basic building block of a wireless sensor node. The *American Heritage[®] Dictionary of the English Language* defines mote as “A very small particle; a speck”. The wireless sensor nodes are expected to reduce to the size of a grain of sand or a dust particle, and hence the name. However, presently the commonly used motes are the size of two size AA batteries. A mote has ADC, MCU, and Radio interface, though it may not have a sensor of interest, e.g., strain gauge in our case. In case it has the required sensor, it may also be called sensor node.

For this project, the hardware and software are developed for Tmote Sky, a mote from Moteiv Corporation. Tmote Sky is a mote platform for extremely low power, high data-rate, sensor network applications designed with the dual goal of fault tolerance and development ease. Tmote Sky has the largest on-chip RAM size (10kB) of any mote, an IEEE 802.15.4 radio, and an integrated on-board antenna providing up to 125 meter range. Tmote Sky offers a number of integrated peripherals including a 12-bit ADC and DAC, Timer, I2C, SPI, and UART bus. Toward development ease, Tmote Sky provides an easy-to-use USB protocol for

programming, debugging and data collection [7, 8] along with a common 10-pin external connector. Some of the key features of Tmote Sky are:

- 8MHz Texas Instruments MSP430 microcontroller (10k RAM, 48k Flash)
- Integrated ADC, DAC, Supply Voltage Supervisor, and DMA Controller
- Integrated onboard antenna with 50m range indoors / 125m range outdoors
- Ultra low current consumption (For MSP430: Sleep mode - $< 3\mu A$, Active mode - $< 600\mu A$)
- Fast wakeup from sleep ($< 6\mu s$)
- Programming and data collection via USB

4.1 Analog to Digital Converter

The superior (in comparison to Mica2, which provides a 10-bit ADC) 12-bit ADC available with Tmote Sky is one of the main reasons of using Tmote for this project. The microcontroller MSP430 on-board Tmote Sky has a module of 12-bit ADC, with reference generator, and a 16 bit conversion-and-control buffer. The conversion-and-control buffer allows up to 16 independent ADC samples to be converted and stored without any CPU intervention. The reference voltage can be selected to be either 1.5 V or 2.5 V, through software. We use the reference voltage of 2.5 V. There are eight individually configurable external input channels, and independent channel-selectable reference sources for both positive and negative references. When the ADC is not actively converting, the core is automatically disabled and automatically re-enabled when needed. The oscillator for ADC is also automatically enabled when needed and disabled when not needed.

4.2 Micro Controller

The microcontroller used in Tmote Sky is MSP430f1611, which is a part of MSP430 family. MSP430 has a 16-bit RISC processor, and memory mapped analog and digital peripherals. It has ultra-low power architecture and consumes less than $600\mu A$ even when active. Its high performance 12-bit ADC, capable of 200,000 samples per second, is ideal for precision measurement. MSP430f1611 has two USART modules capable of operating in UART, SPI & I²C mode. The USART0 is used in this project as UART0 with two pins URXD and UTXD, while USART1 is used as UART1 to communicate with the PC. More information on MSP430 is available in [9].

4.3 Radio Interface

The Chipcon CC2420 radio is a low-cost, highly integrated solution for robust wireless communication in the 2.4 GHz unlicensed ISM band. The CC2420 is an IEEE 802.15.4 compliant radio providing the PHY and some MAC functions. It is highly configurable for many applications with the default radio settings. CC2420 includes a digital direct sequence spread spectrum baseband modem providing a spreading gain of 9 dB and an effective data rate of 250 kbps. It provides extensive hardware support for packet handling, data buffering, burst transmissions, data encryption, data authentication, clear channel assessment, link quality indication and packet timing information. The configuration interface and transmit/receive FIFOs of CC2420 are accessed via an SPI interface. The CC2420 is controlled by the TI MSP430 microcontroller through the SPI port and a series of digital I/O lines and interrupts. The radio may be shut off by the microcontroller for low power duty cycled operation. The CC2420 has programmable output power. More information on CC2420 can be found here [10].

5 Software

The software for this project can be divided in two parts: one resides on the motes, and the other is the software on PC required to interface with the motes.

5.1 Software on the Motes

We use TinyOS, an open source operating system for wireless embedded sensor networks, for the software on mote. TinyOS features a component-based architecture which enables rapid innovation and implementation while minimizing code size as required by the severe memory constraints inherent in sensor networks [11]. The programming language of TinyOS is a modified version of C that uses a custom compiler 'NesC'. Hence, the code for this project has also been written in NesC. More information on TinyOS and NesC can be found at the documentation section of [11].

The software on Mote can be largely divided into two matching parts: software for communicating with the signal conditioning circuit, and software for radio communication. Both of them are treated separately in next two sections.

5.1.1 Software for Signal Conditioning Circuit

The main function of this software is to allow communication between the mote and the PGA, and also to read the output of the signal conditioning circuit using the ADC. The mote and the PGA communicate using one-wire UART compatible interface available on PRG pin at PGA, and the UART0 of the mote. The PGA has to be initially configured by setting its registers as described in section 3.2.1. Proper gain and offset has to be set by writing to appropriate registers. The command for writing or reading a register involves sending a sequence of bytes including the address of the register and the data (in case of writing). Two arrays, *writePGABuffer* and *readPGABuffer* have been created to hold the command for writing or reading from a register respectively. In case of *writPGABuffer*, the data part is modified as necessary. The following functions have been written to control the PGA:

- *void sendOneCommand(char* bufferName, int offset, int commandLength)*: Sends one command over the UART0 to the PGA. The command could be (a) write PGA register (b) read PGA register (c) write to EEPROM or (d) read from EEPROM.

pre-condition: A reply from any other previous command should not be expected from the PGA. The UART must not be transmitting a byte, if it is, the function will not transmit until the previous byte has been sent completely.

post-condition: A flag is set so that any other function doesn't transmit, and a reply can be received without interference; if a reply is not expected, the flag should be reset immediately by the calling function.

@param **bufferName* the command buffer i.e., *writePGABuffer* or *readEEPROMBuffer* etc.;

@param *offset* the address/register to which the command is written, i.e., the row to be sent;

@param *commandLength* the length of the command in the buffer (write and read commands are of different length), i.e., the no. of columns in the buffer;

- *char* readRegister(int regNum)*: Read Register from the PGA. Reads the specified register from the PGA and returns the value. It calls *sendOneCommand()*, handling pre- and post-condition;

@param *regNum* the integer address of the register;

@return the array containing two bytes read from PGA from the specified register;

- *char* readTemperature()*: Read temperature from the PGA. It reads Register 0 from PGA and returns the value;

@return the array containing two bytes read from PGA from Register 0;

- *void setGain(float totalGain)*: Sets the total gain of the PGA. Based on the total gain, it calculates the Front End Gain, Fine Gain and Output Gain, and writes the values in the relevant registers. The algorithm tries to set the maximum possible Front End Gain, then the maximum Output Gain, and finally sets the Fine Gain to make the Total Gain closest possible to the desired gain.

@param *totalGain* The desired total gain of the PGA;

- *float* readGain()*: Read the total gain of the PGA by reading relevant registers.
 @return Returns an array of floats with all gains of PGA in the following order:
 - (0) Total Gain
 - (1) Front End Gain
 - (2) Fine Gain
 - (3) Output Gain
- *void setOffset(float vTotalOffset)*: Sets the overall Total Offset for the PGA (in Volts). Based on the Total Offset, and the current gain, it calculates the Coarse Offset and the Fine Offset for the PGA.
 @param *vTotalOffset* The Total Offset for the PGA.
- *float* readOffset()*: Reads the Total Offset (in Volts) by reading appropriate registers and reading the current gain.
 @return Returns an array of floats containing various offsets of the PGA in the following order:
 - (0) Total Offset Voltage
 - (1) Fine Offset or Zero DAC Voltage
 - (2) Coarse Offset Voltage
- *void programPGA()*: Program the PGA by writing all the registers with current values.

5.1.2 Software for Radio Communication

The motes which are assigned the task of sensing (measuring the strain), have to be controlled and coordinated by a central base station. The base station, during start up, sends 'Command' messages to configure and calibrate the motes and start the application on the motes. The application on the remote motes periodically reads the ADC and sends back 'Data' messages back to the base station.

The purpose of the command messages is to configure the PGA and the signal conditioning circuit, eventually starting the sensing application. The data messages are sent periodically by each of the motes and they contain the ADC value, from which the strain is calculated. These messages are in the TOSMsg (Standard TinyOS Message) format. The customized packet is encapsulated in the TOSMsg. They are sent from the PC to the motes via the base station and vice-versa. The software on the PC is explained in the next section.

The following structures form the command and data messages respectively.

The command message structure is:

```
typedef struct ADCMsg {
    int8_t seqno;
    int8_t action;
    uint16_t source;
    uint8_t hop_count;
    uint8_t time;
    uint16_t setting_val;
} ADCMsg;
```

The data message structure is

```
typedef struct Volt_Msg {
    uint16_t sourceaddr;
    uint16_t remote_seqno;
    uint16_t type;
    long return_val;
    char log[8];
} Volt_Msg;
```

In the next two sections we describe the software running on the base station and the remote motes respectively.

Software on the Base Station:

The base station software is available as a TinyOS package. The application is called TOSBase.

TOSBase is an application that acts as a bridge between the serial and radio links. TOSBase will copy its compiled-in group ID to messages moving from the serial link to the radio, and will filter out incoming radio messages that do not contain that group ID. TOSBase includes queues in both directions, with a guarantee that once a message enters a queue, it will eventually leave on the other interface. The queues allow the TOSBase to handle load spikes more gracefully. TOSBase acknowledges a message arriving over the serial link only if that message was successfully enqueued for delivery to the radio link.

Software on the Remote Motes:

The received commands are processed using a *switch* statement with the *action* field of the *command* message as an argument. The following commands can be sent by the user:

- Program PGA: This option calls the *programPGA()* function of Section 5.1.1
- Set Gain: This option calls the *setGain(float totalGain)* function of Section 5.1.1. The argument which is passed into the function is sent using the *setting_val* field in the command message. This value is set type-casted as a positive integer before creating the command packet.
- Set Offset: This option calls the *setOffset(float vTotalOffset)* function of Section 5.1.1. The argument is sent using the *setting_val* field in the command message. The offset value is type-casted as a positive integer by multiplying by 10000 to accommodate enough decimal precision. The value is sent as *uint_16t* over radio and converted back to float before being used in the *setOffset(float vTotalOffset)* function
- Read ADC: There are two options that are differentiated using the *setting_val* field as a flag.

The first option, uses the *RepeatVoltTimer* (a Repeat Timer) is used to periodically read the ADC values (which will give the corresponding Strain). These values are sent to the base station using the *Data Message*. The timer granularity has been set to 100ms. The *time* field in the *ADCMsg* structure allows the user to set the periodicity. If the value is set to 1, the ADC is queried every 100ms. Currently, we are buffering 20 readings, averaging the value and sending a single *Data Message* with the averaged value. This is the default option.

The second option, uses the *SingleVoltTimer*, a One-Shot Timer. For this option 20 readings are taken, averaged and sent (via a Data Message to the base station). This option is used for *Zero Calibration* and *Span Calibration*.

The Zero Calibration sets the *Offset Voltage* such that the ADC value reads 1.1 volts and results in a near zero strain reading. The offset is fine tuned until the ADC reads approximately 1.1 volts. Zero Calibration is used initially when no load/strain is applied on the specimen.

The Span Calibration is used at the other end of the spectrum, i.e., for high loads; the strain is read from a calibration device and is inserted as a parameter for Span Calibration to calculate the actual gain. It is this gain that is used for future strain readings.

- Timer Stop: This option stops the *RepeatVoltTimer*.
- Mote Reset: This option resets the mote to the preset conditions. This function is for debugging. We are also using this function along with the *Bus Arbitration* mechanism to shift the control between UART0 and Radio.

There are a new set of functions which have been implemented to use the *readRegister*, *readTemperature*, *readGain* and *readOffset* functions explained in Section 5.1.1, format the returned values and send them back to the base stations as Data Messages. These functions are currently not usable because of existing problems with the *Bus Arbitration*.

There is a particular issue with Tmote Sky motes that the UART0 and the Radio share the same hardware and therefore both cannot be accessed at the same time. *BusArbitration* is needed to switch between UART0 and the Radio. The mote should be able to get the command messages over the radio, switch to writing to the PGA using UART0 and immediately switch back to listening over the radio for the next command message.

5.2 Software on the Monitoring Station (PC)

The Command/Data messages are sent/received on the PC via the base station. The software on the PC is built using Java. A *Strain Measurement Tool*, is used to feed in default parameters for the gain, offset etc., and then send the Command Messages. The received Data Messages are processed and the *Voltage* and *Strain* values are displayed.

The Strain Measurement Tool works in conjunction with two standard tools, *Serial Forwarder* and *Message Interface Generator (MIG)*, provided in the TinyOS package. The interface of the strain measurement tool is shown in Fig 6.

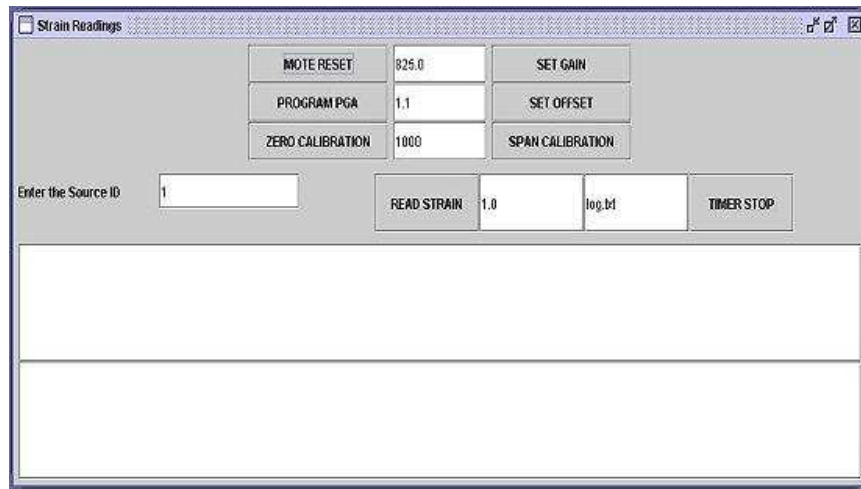


Figure 6: Snapshot of the Java based Strain Measurement Tool

The **SerialForwarder** program is used to read packet data from a serial port and forward it over an Internet connection, so that other programs can be written to communicate with the sensor network over the Internet. To run the serial forwarder, run the following commands:

```
cd tools/java
```

```
java net.tinyos.sf.SerialForwarder -comm serial@COMxx: baud rate
```

The *-comm* argument tells SerialForwarder to communicate over serial port COMxx (belonging to the base station). The *-comm* argument specifies where the packets, SerialForwarder should forward come from.

The *baud rate* argument configures SerialForwarder to communicate at the specified baud rate.

MIG (Message Interface Generator) is a tool that is used to automatically generate Java classes that correspond to Active Message types that is used in the mote applications. MIG reads in the nesC struct definitions for message types in the mote application and generates a Java class for each message type that handles the details of packing and unpacking fields in the message's byte format. MIG is used in conjunction with the *net.tinyos.message* package, which provides a number of routines for sending and receiving messages through the MIG-generated message classes.

MIG is used to create the *ADCMsg.java* and *Volt_Msg.java* classes from the *ADCMsg* and *Volt_Msg* structures respectively. MIG also provides functions, to write/read values to/from every member of the structure. For example, the function *set_time(10)*, sets the granularity of the RepeatVoltTimer to 1 second.

MoteIF represents a Java interface for sending and receiving messages to and from motes. The wrapper java code uses the above mentioned classes and its standard functions along with MoteIF to send commands and receive data.

For more details on the Serial Forwarder, MIG and MoteIF, refer to [12].

Each of the buttons provided on the tool, creates a Command Message, uses the MoteIF interface to send the packet to the base station. The Base station uses the Radio to send messages to the remote motes. There is a one-to-one correspondence between the buttons on the tool, which send the commands and the functions implemented on the remote motes, to process the commands. The latter part has been covered in detail in Section 5.1.2.

The *Strain Measurement Tool* tool offers the following commands:

- *MoteReset*
- *ProgramPGA*
- *SetGain* and a Text Field to input the value to be set.
- *SetOffset* and a Text Field to input the value to be set.
- *Zero Calibration*
- *Span Calibration* and a Text Field to input the value to be set.
- *Read Strain* , a Text Field to set the granularity and another to specify the file name where the readings have to be logged.
- *Timer Stop*
- *Enter Source* and a Text Field to specify the Source ID. This is useful in setting different parameters for each of the remote motes.

Dynamic Graphical Display using Oscilloscope:

TinyOS provides a standard oscilloscope application which graphically displays the data coming from the motes. This will display a window containing a graphical display of the strain readings from the mote. It connects to the serial forwarder over the network and retrieves packet data, parses the sensor readings from each packet, and draws it on the graph.

We have modified the code provided by TinyOS to read accept the *Volt_Msg* structure and extract the sampled ADC value from it. This ADC value is later converted to appropriate *strain* values and plotted in real time.

6 Validation

To validate our calibration procedure and test the accuracy of the resulting system we used two strain gauges mounted side-by-side on a steel bar. We used a 220 kips MTS closed-loop compression machine to control the amount of force applied to the steel bar.

We connected one strain gauge to a portable strain indicator (Vishay Model P3500) and the other strain gauge to the sensor node that was to periodically sample the strain and report to the monitoring station (another mote connected to a laptop). The validation setup is depicted in Fig. 7. Twenty readings are taken at 100ms intervals, averaged and sent to the monitoring station (thus one average reading arrived every 2s). The communication range was impressive (≈ 50 m) despite significant electromagnetic noise in the lab and the lack of line of sight. Therefore we used a single hop although we previously demonstrated multihop capabilities.

Using the software on the monitoring station, we calibrated the mote setup based on the results from the strain indicator (at zero and $1000\mu E$) and compared 40 readings (for each strain setting) of the wireless sensor setup with the ones from the strain indicator. The results are shown in Fig. 8. It is clear that the precision of our setup is comparable with the one of the validation instrument.

7 Future Work

In this section we outline the future work pertaining to this project.

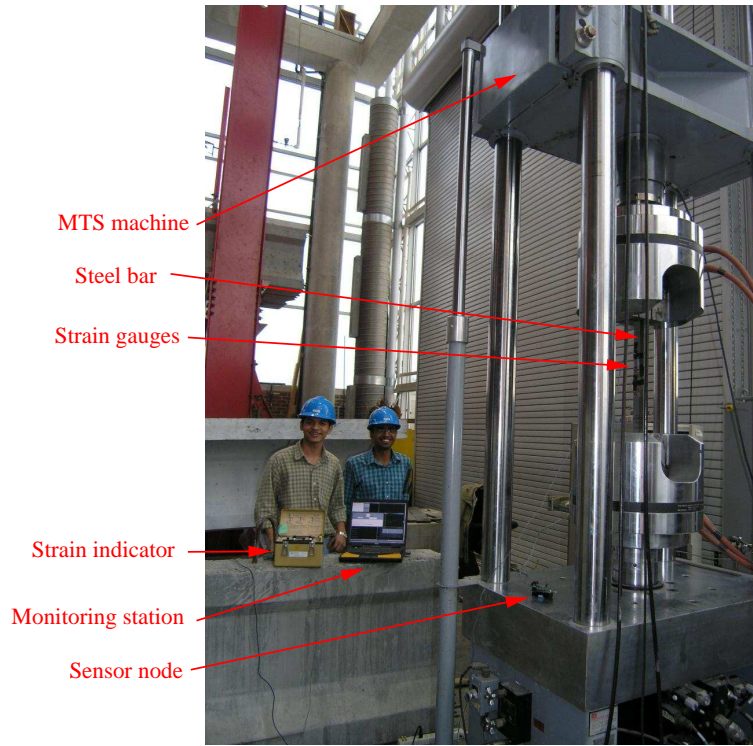


Figure 7: Validation setup.

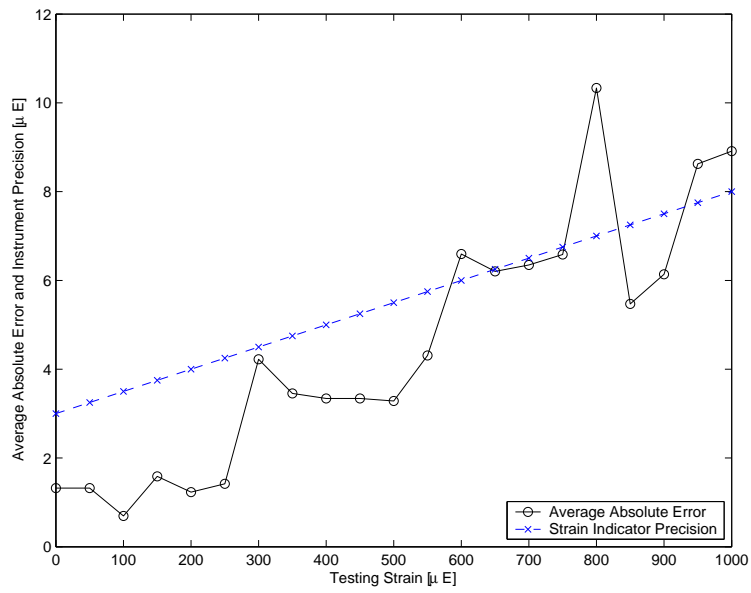


Figure 8: Average absolute errors between the mote setup and the strain indicator.

7.1 Hardware

- Add a jumper to bypass the power control through the CMOS gates.
- Make the labels on the PCB more clear, give number/name to jumper positions.
- Make the drill holes for bridge resistors bigger to let the pins of precision resistors go through.
- Remove the ad-hoc jumper used to bypass the power control, when either the EEPROM or the radio problem is fixed through the software.

7.2 Software

- Use *Bus Arbitration* to be able to use the radio and UART0 without having to reset the mote.
- Write a function to calculate the checksum for EEPROM, and write the settings to the EEPROM.

Glossary

- *ADC*: Analog to Digital Converter, converts analog signal like voltage to digital form which can be read and processed by MCU or any other computer.
- *Axial Strain*: Strain along the length of the element, as opposed to *transverse strain* which is along the breadth of the element.
- *MCU*: Micro Controller Unit, is the central processing unit of the mote. It processes the data locally, can store data in limited memory and can interface with other devices.
- *Mote*: The basic building block of a wireless sensor node, it has ADC, MCU, and Radio interface, though it may not have a sensor of interest, e.g. strain gauge in our case. In case it has the required sensor, it may also be called sensor node.
- *OP-AMP*: Operational Amplifier, a general purpose amplifier usually used as a building block of instrumentation amplifier.
- *PGA*: Programmable Gain Amplifier, the gain and offset for the amplifier can be programmed dynamically, without any change in the hardware.
- *Signal Conditioning Circuit*: Takes the raw signal from the sensor and prepares it for further processing by amplification, linearization, filtering, etc. The output of the signal conditioning circuit is ready to be read by MCU using ADC.
- *Strain Gauge*: A sensor designed specifically to measure mechanical strain. It is basically a resistive element whose resistance changes with the applied strain.
- *Wireless Sensor Network*: An (ad-hoc) wireless network formed by sensor nodes and/or motes.
- *Wireless Sensor Node*: A mote along with the necessary sensor and interfacing circuit (if necessary).

References

- [1] A. S. Khan and X. Wang, *Strain Measurements and Stress Analysis*. Prentice Hall, NJ, 2001.
- [2] “PGA309 Datasheet - Voltage Output Programmable Sensor Conditioner.” <http://focus.ti.com/lit/ds/symlink/pgs309.pdf>, January 2005.
- [3] “PGA309 User’s Guide.” <http://focus.ti.com/lit/ug/sbou024a/sbou024a.pdf>, January 2005.

- [4] “24AA16/24LC16B Data Sheet - 16 Kbit Serial Interface EEPROM.” <http://ww1.microchip.com/downloads/en/DeviceDoc/21703E.pdf>, 2005.
- [5] “Analog Filter Wizard.” <http://www.analog.com/Wizard/filter/filterUserEntry/>.
- [6] “AD822 Data Sheet - Single Supply, Dual Precision, Rail to Rail Low Power FET-Input Op Amp.” http://www.analog.com/UploadedFiles/Data_Sheets/146518556AD822_e.pdf, January 2003.
- [7] “Tmote Sky.” <http://www.moteiv.com/products.php>.
- [8] “Tmote Sky Datasheet.” <http://www.moteiv.com/products/docs/tmote-sky-datasheet.pdf>, March 2005.
- [9] “MSP430x1xx User’s Guide.” <http://www-s.ti.com/sc/psheets/slau169/slau169.pdf>, September 2005.
- [10] “CC2420 Radio Data Sheet Revision 1.3.” http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf, October 2005.
- [11] “TinyOS website.” <http://tinyos.net/>.
- [12] “TinyOS Tutorial.” <http://www.tinyos.net/tinyos-1.x/doc/tutorial/lesson6.html>.

A Part List

Following is a list of parts for the signal conditioning circuit. The approximate cost of all parts for one circuit is \$60 (less if bought in bulk). The most expensive components are the bridge completion resistors (\$45), the programmable amplifier (\$7) and the operational amplifier (\$6).

| Item | Description |
|------------------|---|
| PGA309 | Programmable Gain Amplifier; TSSOP pkg; PGA309AIPWT; TI; U2 |
| AD822AN | OP-AMP Filter; DIP; Analog Devices; U1 |
| 24LC16B | 16Kbit EEPROM; PDIP; Microchip; U3 |
| SN74HC04 | CMOS Inverter; Hex; TI; U4 |
| 56.2 K Ω | Resistor 56.2 K Ω ; 1% tolerance; TCR 100 ppm/C; 0.125W; R8 & R9 |
| 25.5K Ω | Resistor 25 K Ω ; 1% tolerance; TCR 100 ppm/C; 0.125 W; R10 |
| 100 nF | Capacitor 100nF (0.1 μ F); 5% tolerance; C4; Size 0805 |
| 47 nF | Capacitor 47nF (47000 pF); 5% tolerance; C3; Size 1206 |
| 100 Ω | Resistor 100 Ω ; 1% tolerance; TCR 100 ppm/C; 0.125W, R6 & R7 |
| 10 nF | Capacitor 10 nF; 5% tolerance; C2; Size 1206 |
| 150 pF | Capacitor 150 pF; 5% tolerance; C1; Size 0805 |
| 1K Ω | Foil resistor 1K Ω ; 0.01% tolerance; Vishay S102C; R4 |
| 5K Ω | Foil resistor 5K Ω ; 0.01% tolerance; Vishay S102C; R1 & R3 |
| strain connector | 3 connector terminal block |
| mote connector | Double row header |
| mote connector | Double row socket |