Tran-SET

Transportation Consortium of South-Central States

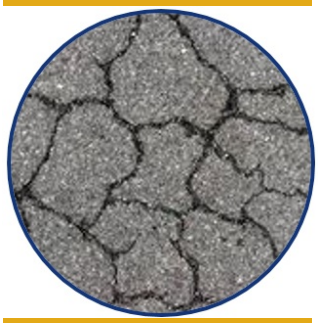
Solving Emerging Transportation Resiliency, Sustainability, and Economic Challenges through the Use of Innovative Materials and Construction Methods: From Research to Implementation

Development of a Self-Powered Structural Health Monitoring System for Transportation Infrastructure

Project No. 17PTAM03

Lead University: Texas A&M University

Collaborative Universities: University of Texas at San Antonio



Enhancing Durability and Service Life of Infrastructure

Final Report

December 2018

Disclaimer

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated in the interest of information exchange. The report is funded, partially or entirely, by a grant from the U.S. Department of Transportation's University Transportation Centers Program. However, the U.S. Government assumes no liability for the contents or use thereof.

Acknowledgments

The authors would like to acknowledge the support by the Transportation Consortium of South-Central States (Tran-SET).

TECHNICAL DOCUMENTATION PAGE

1. Project No.	2. Government Accession No.	3. Recipient's Catalog No.
17PTAM03		

4. Title and Subtitle	5. Report Date
Development of a Self-Powered Structural Health Monitoring System for Transportation Infrastructure	Dec. 2018 **6. Performing Organization Code**
7. Author(s) PI: Aydin I Karsilayan https://orcid.org/000-0001-8694-8836 Co-PI: Samer Dessouky https://orcid.org/0000-0002-6799-6805 Co-PI: Athanassios T. Papagiannakis https://orcid.org/0000-0002-3047-7112	**8. Performing Organization Report No.**
9. Performing Organization Name and Address Transportation Consortium of South-Central States (Tran-SET) University Transportation Center for Region 6 3319 Patrick F. Taylor Hall, Louisiana State University, Baton Rouge, LA 70803	**10. Work Unit No. (TRAIS)** **11. Contract or Grant No.** 69A3551747106
12. Sponsoring Agency Name and Address United States of America Department of Transportation Research and Innovative Technology Administration	**13. Type of Report and Period Covered** Final Research Report May 2017 – May 2018 **14. Sponsoring Agency Code**

15. Supplementary Notes

Report uploaded and accessible at: Tran-SET's website (http://transet.lsu.edu/)

16. Abstract

Roadways and bridges play an important role in the economic and social health of society by connecting commerce and people. Economic growth and population expansion pose considerable burden on the aging infrastructure (i.e., pavements and bridges). There is a pressing need to develop structural health monitoring (SHM) technologies capable of collecting infrastructure utilization data. Doing so inexpensively with self-powered systems will revolutionize infrastructure monitoring technology, and will improve decision making enabling roadway and bridge preservation. In this study, a self-powered battery-less structural health monitoring (SHM) system was developed. It is powered by a thermal energy harvester equipped with thermoelectric generators (TEGs) driven by temperature differentials between the top of asphalt pavements and their lower layers. An innovative 2-tier TEG harvester was designed to limit the downtime of the SHM system when the temperature differentials are insufficient to power a single unit. The 2-tier system requires a minimum of 2.1^{o}C in temperature differential to generate the minimum of 40 mV needed to power the SHM system. The SHM system consists of a DC-DC booster to increase the voltage generated by the harvester, a buck controller to bring this voltage down to the 3.3 Volts required for powering the microcontroller, a microcontroller and a wireless transceiver for transmitting the data. Another transceiver carried on-board a pilot vehicle is needed to retrieve the data. Software was developed in this project to allow data communication between the two transceivers. The SHM system developed accepts analogue voltage input from any sensor that generates analogue voltage, (e.g., piezoelectric axle load sensors, strain gauges, temperature gauges and so on). A prototype of this SHM system was constructed and tested in the lab and it is ready for field implementation.

17. Key Words	18. Distribution Statement
Structural Health Monitoring, Power Harvesting, Sensor, Thermoelectric	No restrictions.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	38	

Form DOT F 1700.7 (8-72) **Reproduction of completed page authorized.**

SI* (MODERN METRIC) CONVERSION FACTORS

APPROXIMATE CONVERSIONS TO SI UNITS

Symbol	When You Know	Multiply By	To Find	Symbol
		LENGTH		
in	inches	25.4	millimeters	mm
ft	feet	0.305	meters	m
yd	yards	0.914	meters	m
mi	miles	1.61	kilometers	km
		AREA		
in^2	square inches	645.2	square millimeters	mm^2
ft^2	square feet	0.093	square meters	m^2
yd^2	square yard	0.836	square meters	m^2
ac	acres	0.405	hectares	ha
mi^2	square miles	2.59	square kilometers	km^2
		VOLUME		
fl oz	fluid ounces	29.57	milliliters	mL
gal	gallons	3.785	liters	L
ft^3	cubic feet	0.028	cubic meters	m^3
yd^3	cubic yards	0.765	cubic meters	m^3
		NOTE: volumes greater than 1000 L shall be shown in m^3		
		MASS		
oz	ounces	28.35	grams	g
lb	pounds	0.454	kilograms	kg
T	short tons (2000 lb)	0.907	megagrams (or "metric ton")	Mg (or "t")
		TEMPERATURE (exact degrees)		
^oF	Fahrenheit	5 (F-32)/9 or (F-32)/1.8	Celsius	^oC
		ILLUMINATION		
fc	foot-candles	10.76	lux	lx
fl	foot-Lamberts	3.426	$candela/m^2$	cd/m^2
		FORCE and PRESSURE or STRESS		
lbf	poundforce	4.45	newtons	N
lbf/in^2	poundforce per square inch	6.89	kilopascals	kPa

APPROXIMATE CONVERSIONS FROM SI UNITS

Symbol	When You Know	Multiply By	To Find	Symbol
		LENGTH		
mm	millimeters	0.039	inches	in
m	meters	3.28	feet	ft
m	meters	1.09	yards	yd
km	kilometers	0.621	miles	mi
		AREA		
mm^2	square millimeters	0.0016	square inches	in^2
m^2	square meters	10.764	square feet	ft^2
m^2	square meters	1.195	square yards	yd^2
ha	hectares	2.47	acres	ac
km^2	square kilometers	0.386	square miles	mi^2
		VOLUME		
mL	milliliters	0.034	fluid ounces	fl oz
L	liters	0.264	gallons	gal
m^3	cubic meters	35.314	cubic feet	ft^3
m^3	cubic meters	1.307	cubic yards	yd^3
		MASS		
g	grams	0.035	ounces	oz
kg	kilograms	2.202	pounds	lb
Mg (or "t")	megagrams (or "metric ton")	1.103	short tons (2000 lb)	T
		TEMPERATURE (exact degrees)		
^oC	Celsius	1.8C+32	Fahrenheit	^oF
		ILLUMINATION		
lx	lux	0.0929	foot-candles	fc
cd/m^2	$candela/m^2$	0.2919	foot-Lamberts	fl
		FORCE and PRESSURE or STRESS		
N	newtons	0.225	poundforce	lbf
kPa	kilopascals	0.145	poundforce per square inch	lbf/in^2

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS, ABBREVIATIONS, AND SYMBOLS

A/D	Analog/Digital
BLE	Bluetooth Low Energy
CPU	Central Processing Unit
DC	Direct Current
GB	Giga Bytes
I/O	Input/Output
kb	Kilobits
kbps	Kilobits per second
LDO	Low-Dropout
MB	Mega Bytes
PCB	Printed Circuit Board
RF	Radio Frequency
SHM	Structural Health Monitoring
TAMU	Texas A&M University
TEG	Thermoelectric Generator
USB	Universal Serial Bus
UTSA	University of Texas at San Antonio

EXECUTIVE SUMMARY

This report describes the development of a self-powered battery-less structural health monitoring (SHM) system capable of processing analog voltage input from a variety of sensors, such as strain gauges, piezoelectric weighing strips, and so on. It is powered by an energy harvester driven by thermoelectric generators (TEG). TEGs function on the Seebeck/Peltier principal. TGEs translate the thermal differences between the upper and lower layers of asphalt concrete pavements into electrical energy.

The SHM system consists of a TEG harvester connected to a DC-DC booster circuit to increase the voltage from tens of millivolts to 10 volts DC. A switching regulator (buck converter) is used to further adjust this DC voltage to levels suitable for powering a microcontroller and a wireless transceiver (i.e., 3.3 volts). An off-the-shelf low-power microcontroller was used combined with a wireless transceiver. Another transceiver, either stationary or carried on-board a pilot vehicle, is needed to retrieve the data. Software was developed to allow data communication between the two transceivers.

Extensive thermal analysis was carried out in this study to establish the temperature distribution with depth in asphalt pavements. Environmental data from two extreme locations in the US were used as input, namely a site in South Texas and another in Northern Minnesota. This analysis revealed that temperature differentials between the top of the asphalt pavement and the lower layers are significant. Regardless of lower depth location in the pavements, they have a pattern of crossing the zero voltage twice per day. To compensate for this, an innovative 2-tier TEG harvester in parallel was designed to limit the downtime of the SHM system. One TEG harvester is powered by the temperature differential between 0.02 and 0.15 m, while the other is powered by the temperature differential between 0.02 and 0.50 m. This design generated the least required temperature differential of 2.1°C for powering a SHM system (i.e., 40 mV) 95% of the time.

The SHM system developed accepts voltage input from any sensor that generates analogue voltage. This signal can be digitized by the microprocessor and processed through a programmable transfer function. The amount of data generated and stored depends on the quantity being monitored. For example, if the axle loads generated by a piezoelectric sensor are monitored, their peak values may only need to be stored. In this case, storage requirements are minimal, while their wireless retrieval is possible from a moving vehicle.

In this project, a prototype of this SHM system was constructed and tested in the lab. It was successful in storing and transmitting data under TEG harvester power alone. The developed prototype is ready for field implementation.

IMPLEMENTATION STATEMENT

In order to disseminate the outcomes of this project, the following activities were conducted during the implementation phase:

- Two peer-reviewed papers were published,
- Summer Workshop of TRB's ADC60 Committee on Resource Conservation and Recovery was given, and
- Field testing and experimentation was conducted on the University of Texas at San Antonio (UTSA) campus on the system functionality (July 2018).

In addition, the Thermal energy harvester is being co-funded by another grant from the CPS Energy at UTSA. CPS Energy is the public service utility company at San Antonio. For potential climate benefits to reduce pavement temperature, CPS Energy is intersected for sustainable solution towards smart city initiative including solution to Heat Urban Island.

Within this project, only a prototype based on off-the-shelf components were developed. The resulting sensor is far from being a commercial product, which requires more research, development and testing.

1. INTRODUCTION

Roadways and bridges are the most important components of civil engineering infrastructure. They play an important role in the economic and social health of society by connecting commerce and people. Economic growth and population expansion pose considerable burden on the aging infrastructure (i.e., pavements and bridges). There is a pressing need to develop structural health monitoring (SHM) technologies capable of collecting infrastructure utilization data. Doing so inexpensively with self-powered systems will revolutionize infrastructure monitoring technology, and will improve decision making enabling roadway and bridge preservation.

SHM technologies have emerged within the last two decades enabling continuous monitoring of infrastructure using sensors to collect data on critical response parameters *(9, 11, 12, 16).* Recent literature suggests that embedded sensors in infrastructure can provide continuous data on the status of various structural elements (such as pavements and bridges). Although there is a variety of sensors available for infrastructure monitoring, current technologies mostly rely on external electrical power from either electrical power lines or batteries. This hinders their widespread use and makes them prohibitively expensive. Recently introduced energy harvesters show promise in powering SHM systems. They are powered by one of the following energy harvesting technologies:

- Solar involving photovoltaic cells that convert solar radiation into electrical energy,
- Piezoelectric that converts mechanical stress changes to electrical energy, and
- Thermoelectric that converts thermal differentials into electrical energy using thermoelectric generators (TEGs).

Each of these harvesting technologies has distinct advantages and disadvantages. Comparative reviews can be found in the literature *(8, 13).* Thermoelectric energy conversion has several advantages that make it ideal for powering roadside SHM systems. It is not intrusive nor takes physical roadside space being embedded into the pavement layers. It can be installed below the pavement surface at depths that allow pavement rehabilitation without having to be removed and reinstalled. It has no moving parts nor batteries and hence requires no routine maintenance. It can power SHM systems, while there is sufficient temperature differential to power the TEGs.

2. OBJECTIVE

The objective of this project is to develop a self-powered system capable of recording data from structural health monitoring (SHM) sensors, such as traffic counters, axle weighing sensors, strain gauges, etc. The particular design requirements set forth for this system are:

- It accepts analog voltage input form a variety of SHM sensors,
- It is readily programmable to allow processing the voltage input into meaningful SHM parameters such as peak load, peak strain, etc.,
- It functions without a storage battery, and
- It allows for continuous wireless data retrieval.

3. SCOPE

The development of this self-powered system is based on a thermal energy harvester developed in an earlier research study *(7)*. This harvester utilizes the temperature differentials between the surface of asphalt pavements and their lower layers to generate electrical power through thermoelectric generators (TEGs). Surface heat is transferred to the lower layers through insulated copper plates, while at the lower part the harvester is kept cool through a heat sink. TEGs generate power in response to temperature differences, which is referred to as the Seebeck/Peltier principle *(14)*.

The work presented in this report builds on this thermal harvester by adding the hardware and software necessary for conditioning the power being generated, and using it to drive a programmable microprocessor capable of capturing and recording SHM data and transmitting them wirelessly. Doing so under the limited power generated by the TEG harvester without a storage battery is especially challenging.

In this study, two locations in the US were selected to represent extreme weather conditions in the United States, namely San Antonio, Texas and International Falls, Minnesota. Climatic data for each of these locations was obtained from the LTPP database for a period of a full year.

4. METHODOLOGY

4.1. TEG Harvester Development

TEGs work on the Seebeck/Peltier principle *(14)*. This is described as the production of an electromotive force when two dissimilar materials form a loop with different temperatures at each junction. Using this principle, voltage levels exceeding 40mV can be obtained from an individual TEG in response to temperature differentials as low as couple of degrees Celsius.

For the prototype designed, the commercially available TEG TXL-287-03Z *(6)* was used to convert the temperature differentials between the road surface and the lower pavement layers to a usable voltage difference. The open-circuit output voltage (V_{oc}) obtained from one of these TEGs is given by:

$$V_{oc} = N(0.0002 \times 1.004^{\Delta T}) \times \Delta T \qquad [1]$$

where:
N = number of TEG thermoelectric elements, and
ΔT = temperature difference.

The TXL-287-03Z used has 287 thermoelectric elements (Figure 1). It measures 62 mm x 62 mm x 5.3 mm. Connecting these TEGs in series can increase the voltage generated, while connecting them in parallel can increase the current being generated. Of course, doing so increases the overall internal resistance of the harvester. Several harvester prototypes were developed with either 2 or 4 TEGs connected in parallel. Additional design features of the harvester are:

- the copper plates transferring heat from the top of the pavement (0.02 m under the surface) to lower layers, and
- a Heat Sink at the lower layers for dissipating heat keeping the temperature of the lower layers relatively constant.

A schematic of the harvester is shown in Figure 2. Various design features of the TEG harvester were tested using the finite element package Abaqus (Figure 3).

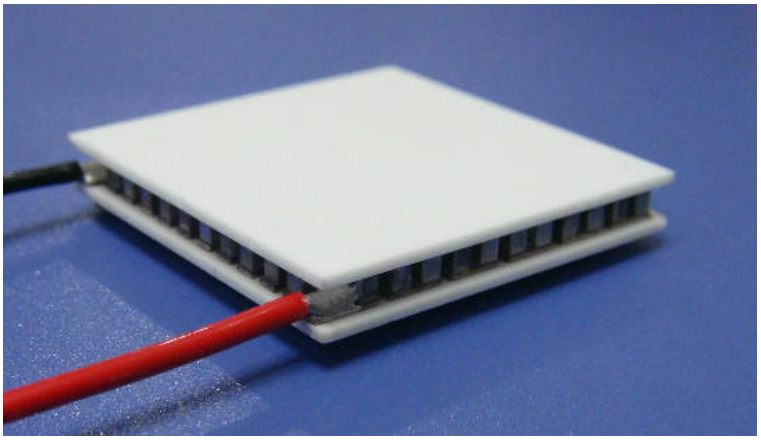


Figure 1. Thermo-electric generator model TXL-287-03 *(6)*.

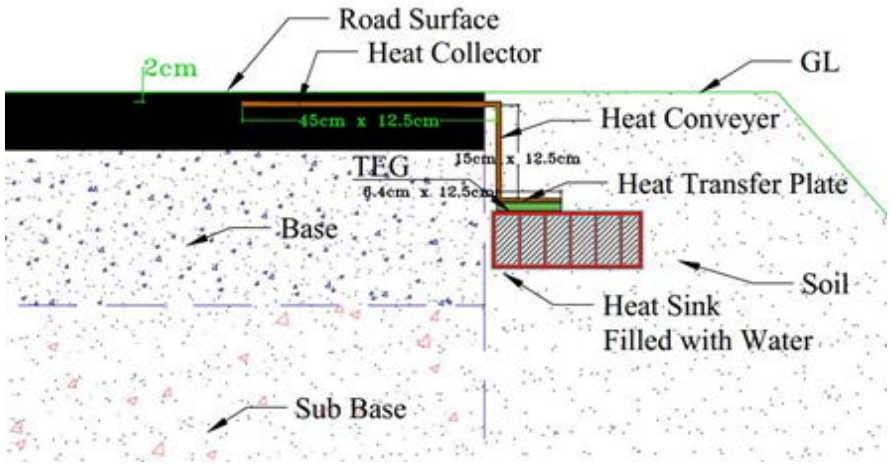


Figure 2. Cross-section of the harvester *(7)*.

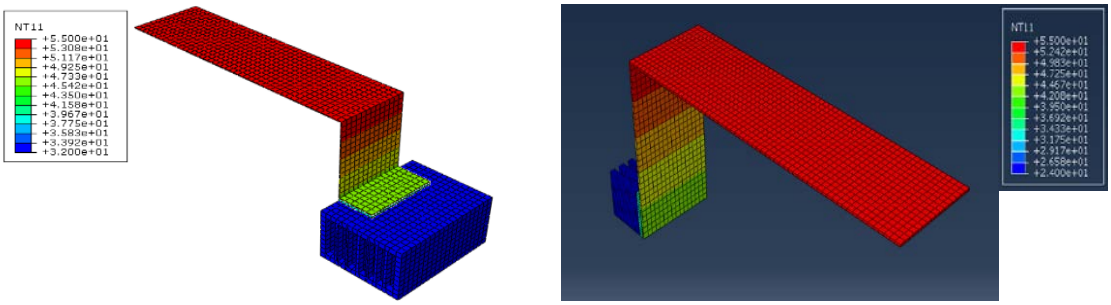


Figure 3. Finite element heat transfer simulations of the harvester *(7)*.

Extensive laboratory testing of several harvester prototypes was conducted independently by the University of Texas at San Antonio (UTSA) and Texas A&M University (TAMU) to verify that the output of the harvester yields usable voltage and amperage levels (Figure 4).

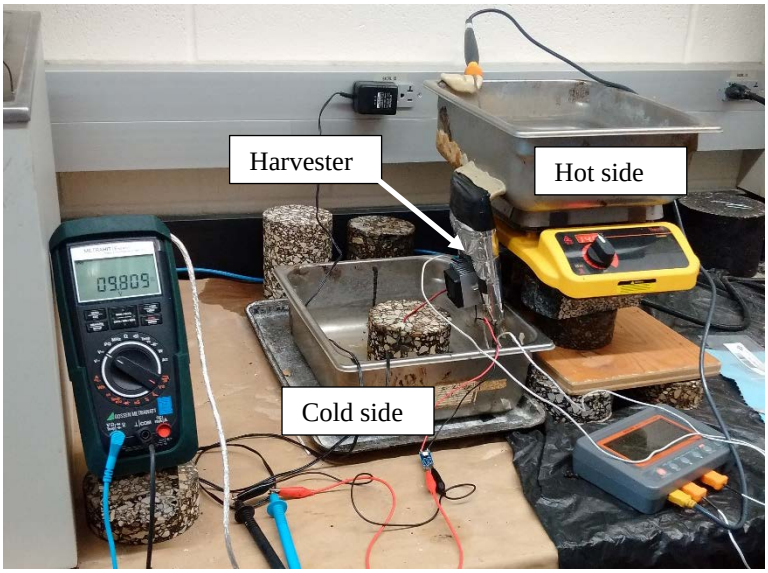


Figure 4. Laboratory testing of the TEG harvester prototype.

4.2. Analysis of Temperature Profiles in Asphalt Pavements

A critical component of the development of the self-powered SHM system was determining the pavement temperature differential fluctuations versus time. The reason was that when the temperature difference ΔT drops below a threshold, the harvester will not generate sufficient power for the SHM system. For this purpose, the computer program TEMPS *(15)* was used in this project to analyze temperature profiles for two locations that represent extreme weather conditions in the United States, namely San Antonio, Texas and International Falls, Minnesota. Weather data for a specific site in each of these locations was obtained from the LTPP database *(10)* for a period of a full year. In the TEMPS analysis of temperature profiles of this study, a pavement structure consisting of 0.15 m of asphalt concrete (AC) layer and 0.40 m of unbound limestone base was considered. The thermal properties of the AC layers and the soil subgrade were obtained from the literature. For the asphalt concrete the coefficients of thermal permeability and thermal capacity used were 910 J/kg/ºK and 1.167 W/m/ºK, respectively. The albedo of the surface was assumed to be 0.15. The computer software TEMPS allowed predicating pavement temperature profiles at hourly increments. This data was analyzed to obtain ΔT estimates between 2 pairs of pavement depth locations:

- Temperature at 0.02 m from the surface minus temperature at 0.15 from surface, and
- Temperature at 0.02 m from the surface minus temperature at 0.50 from surface.

The top location (i.e., 0.02 m) corresponds to the upper location of the heat transfer plate of the harvester, and the lower the location of the heat sink of the harvester. An example of the results is shown in Figure 5. It can be seen that these ΔTs follow a pattern that crosses zero twice per day, once in the morning and the other in the evening. It is observed, however, that these crossings do not happen at the same time for these two sets of locations.

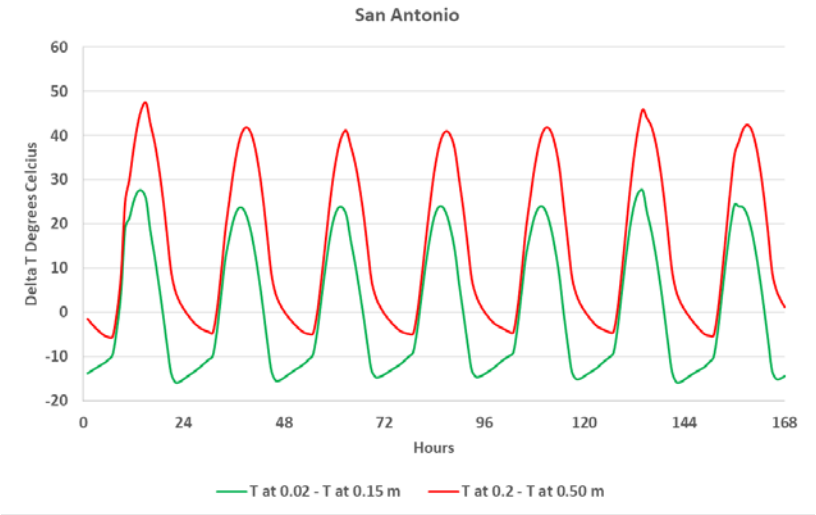


Figure 5. Estimated temperature differentials in asphalt concrete in San Antonio, TX (6/1/2013 – 6/7/2013).

Although brief, these zero temperature crossings will cause interruption to the power of the battery-less SHM system. This problem was resolved by placing two sets of TEG harvesters connected in parallel, one with the heat sink at 0.15 m and the other with a heat sink at a 0.50

m depth. This 2-tier design minimizes the time periods when there is insufficient power to drive the SHM system. Analysis of the time period when insufficient power is available for driving the SHM is presented later.

4.3. Design of the SHM System

The SHM system is powered by the TEG harvester described above. It consists of four main additional components:

- A DC-DC boost converter,
- A buck converter,
- A microcontroller, and
- Wireless transceiver communicating to an external access point.

The function of these components is described in detail below. A block schematic of the components making up the SHM system is shown in Figure 6 enclosed by a red dotted line.

4.3.1. DC-DC Booster and Buck Converter

The voltage generated from the TEG harvester could be as low as a few millivolts. Although sufficient levels of power could be harvested most of the time through the use of multiple TEGs, there is still a need to boost the power to the levels needed to operate a microcontroller. To increase the TEG harvested voltage to higher levels, a commercially-available DC-DC booster (VB-0410-2 Gold) shown in Figure 7 was used in this project *(4)*. This circuit provides approximately 10V DC output for input levels as low as 40mV. The device is built to operate under both positive and negative input voltages and hence can accommodate either the positive or negative voltages generated depending on the sign of the temperature differences powering the TEGs.

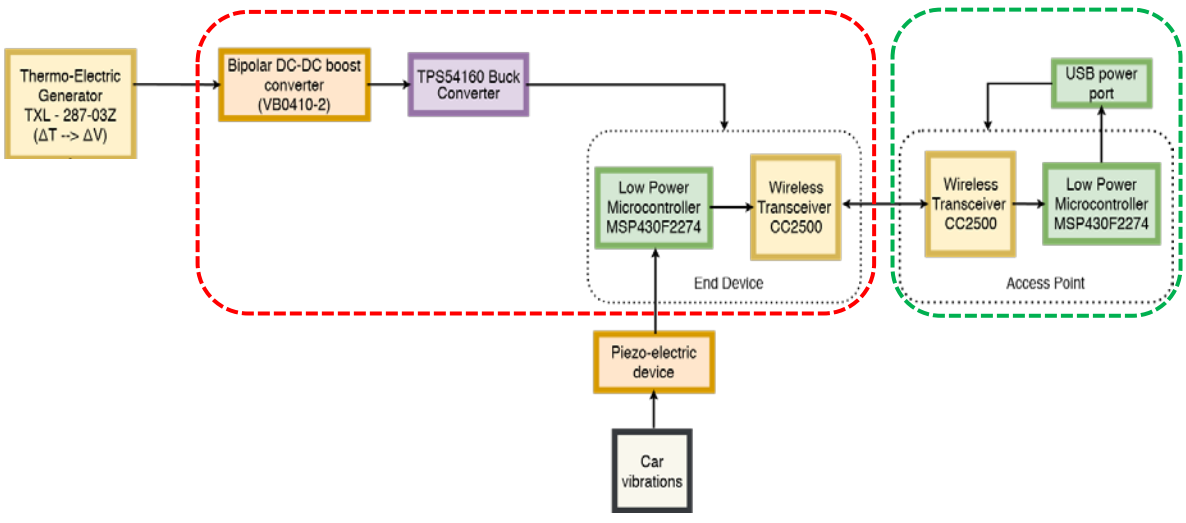


Figure 6. Self-powered SHM system block diagram.

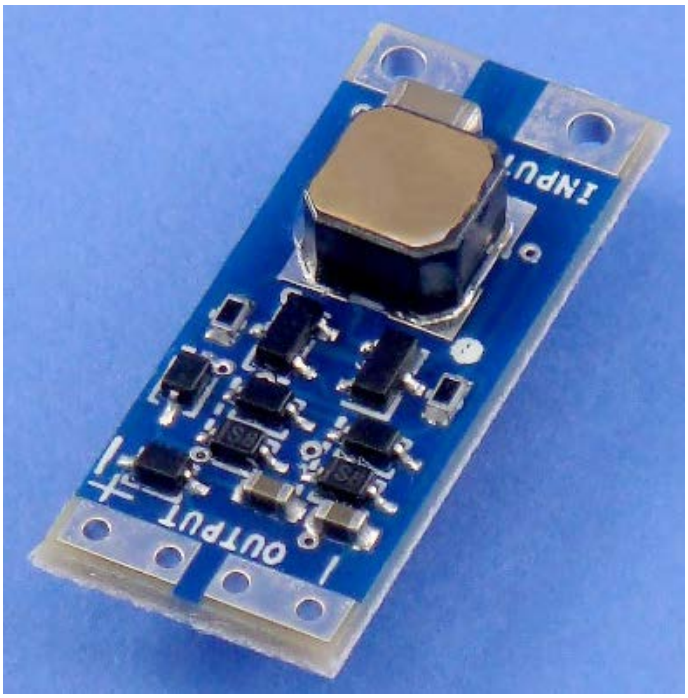


Figure 7. DC-DC Booster VB-0410-2 Gold *(4)*.

The resulting 10V output from the DC-DC booster, however, is too high for the low-voltage/low-power microcontroller circuitry powering commercially available microprocessors, such as those made by Texas Instruments and Arduino. Therefore, an efficient regulator to reduce the 10V DC to 3.3V DC is required. Among all regulators available in the market, low-dropout regulators (LDOs) provide clean DC output but cannot accommodate the large voltage drop required in this case (i.e., from 10 to 3.3 Volts). As an alternative, a switching-mode voltage regulator, specifically a buck converter was used. Buck converters have a high conversion efficiency and deliver an acceptable DC signal quality. For the SHM system prototype, the commercially available buck converter TPS54160 from Texas Instruments *(3)* was used (Figure 8).

Figure 8. Texas Instruments TPS54160 buck converter *(3)*.

4.3.2. Microcontroller and Wireless Transceiver/External Access Receiver

Data acquisition and storage operations are performed by a Texas Instruments MSP430 microcontroller, paired with CC2500 wireless transceiver on the same printed circuit board (PCB) *(5)* (Figure 9). These two elements comprise the End Device in the conceptual diagram shown earlier (Figure 6). Vehicle data can be collected using any commercially available sensor that generates analogue voltage output, such as axle load weighing sensors or strain gauges. An example is the piezoelectric strip sensors used for weighing vehicles in motion (e.g., MSI's *RoadTrax BL* sensor shown in Figure 10). The particular transfer function between the sensor voltage output and the quantity being monitored needs to be established and programmed into the microprocessor. In this case, this transfer function is the relationship between peak voltage and axle load. The quantity to be computed and stored would be the peak load value of each axle.

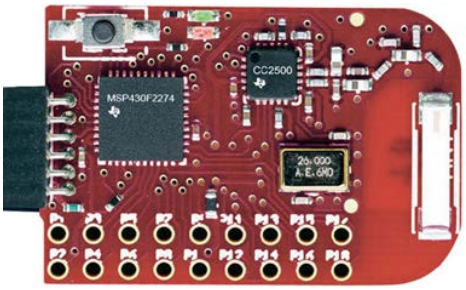


Figure 9. MSP430 microcontroller and CC2500 transceiver *(5)*.

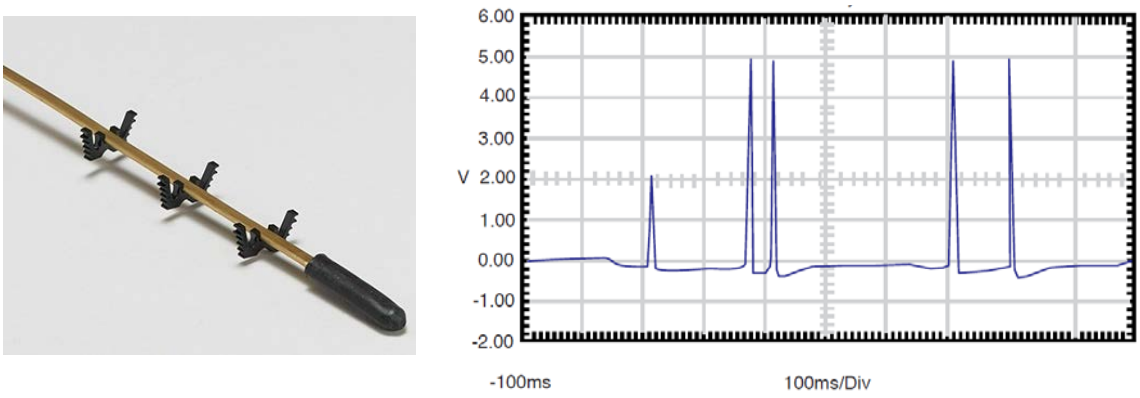


Figure 10. *RoadTrax* Brass Linguini (*BL*) axle sensors and example output from 5-axle truck.

The CC2500 wireless transceiver runs on the *SimpliciTI* protocol *(1)*, which is a low-power radio frequency (RF) network protocol optimized for small-sized RF networks having a limited number of nodes communicating directly to each other or to an external access point through a range extender. The external access point is indicated by a green dotted line in Figure 6. The CC2500 is integrated with a highly-configurable baseband modem. The modem supports several modulation formats and has a maximum configurable data rate of 500 kBaud.

In the prototype system, the Transceiver CC2500 is used together with the MSP430F2274 controller in the End Block, as well as in the Access Point shown in Figure 11. The Access Point collects data from the End Device. A custom code was developed in this study for this purpose (see the Appendix) which can be uploaded to the receiver via the USB port. The end device is to be installed roadside or be carried by a pilot vehicle that collects data by driving in the vicinity of the SHM sensor. When an Access Point is in range of an End Device, the association between two applications, takes place, referred to as linking. The linking process creates a connection between "peers" through which messages can be exchanged. When linking is established, it is always a bi-directional connection, indicated by blinking of the on-board LEDs. In the prototype system, the End Device LED lights were disabled to conserve power. Therefore, the indicator of an established connection is the blinking of the Access Point LEDs only.

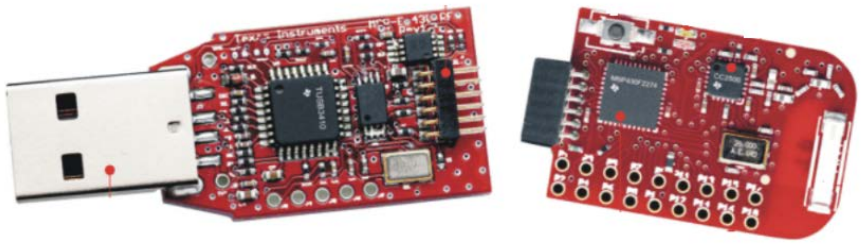


Figure 11. Access point.

The CC2500 transceiver operates in three modes:

1. Demo mode: This is the default mode of operation of the Transmitter on the End Device. While in this mode, the device continuously looks for an Access Point (AP) to

pair with. As soon as an AP is detected, it is ready to transmit all the data it has been acquiring through its inputs and processing through its on-board ADC in the MSP430F2274.

2. Transmit mode: The End Device enters this mode once an AP is detected and ready for data transmission. The End Device will transmit all the digitized data on its memory onto the Access Point as long as it is within range. This mode requires 1 mA at 3 V (i.e., it consumes 3 mW), at a rate of 1.2 kbps. The establishment of a connection between the End Device and the AP is signaled by the blinking of one of the LEDs on the AP. If the connection is lost, the blinking stops and restarts if the connection is restored.
3. Sleep mode: This mode requires the least power. In this mode, the End Device is not actively looking for an AP to pair with. It simply takes in analog data, processes it, and stores it in memory. This mode typically operates at 390µA at 3 V (i.e., it consumes 1.17 mW).

Obviously, the power consumption for the Transmit mode is much higher than that of the Sleep mode even at modest transmission rates. As discussed later, the decision was made to keep the Transmit mode permanently on to allow data uploading any time the AP is nearby.

5. FINDINGS

The functionality of each block shown in Figure 6 was tested by lab measurements. After testing each component separately, the blocks were integrated to test the complete system. At this stage, the system functionality was verified and the prototype was deemed ready for field testing.

5.1. Output of TEG Harvester

Figure 12 shows the measured open-circuit output voltage of a TXL-287-03Z TEG as a function of temperature difference, as well as the calculated voltage based on Equation 1. Experimental results were in agreement with the theoretical calculations. The small errors observed were possibly due to measurement inaccuracies and/or lack of thermometer calibration.

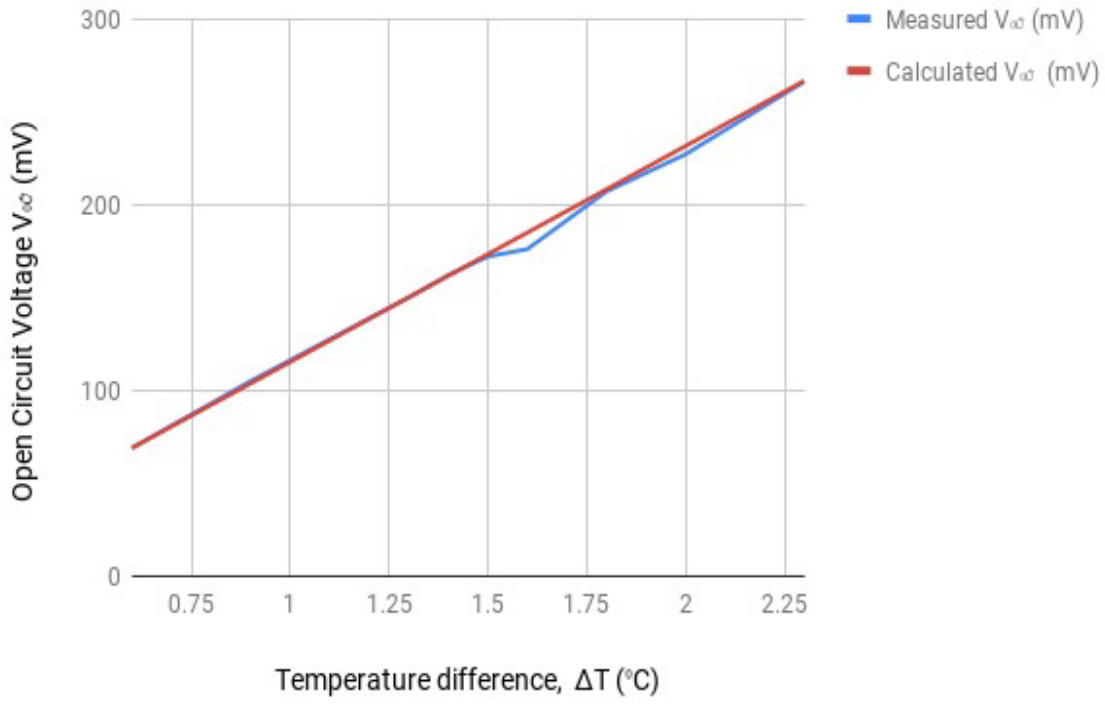


Figure 12. Measured and calculated output voltage of TXL-287-03Z TEG as a function of Δ*T*.

5.2. Required Temperature Differential

5.2.1. Minimum Temperature Differential Needed to Power the DC-DC Booster

Figure 13 shows the estimated and measured output voltage of the VB-0410-2 Gold DC-DC booster circuit as a function of the applied input voltage. The estimates were obtained from the booster's manufacturing specifications. The lab measurements were in close agreement to these estimates. Clearly, the output voltage is approximately 10V for any voltage supplied to the booster above a 40mV threshold. Theoretically, this level of voltage can be generated by a single TEG unit from a mere 0.5°C temperature differential Δ*T* (Figure 12). However, considering the load (i.e., resistance) of the booster, a slightly higher minimum temperature differential would be necessary to generate the minimum 40mV output needed. This minimum

temperature, is referred to as the turn-on differential temperature. To maintain a reasonably low turn-on differential temperature under load, several TEG units connected in parallel are necessary. Figure 14 shows the corresponding turn-on differential temperature as a function of the number of TEG units used per harvester. This plot suggests, for example, that a four-unit TEG harvester is sufficient to maintain operation of the booster at the relatively low turn-on temperature differential of 2.1ºC.

Figure 15 verifies this experimentally by showing that a minimum ΔT of 2.1ºC is sufficient to power the system under electrical load. Figure 16 shows the buck converter output, which provides the supply voltage to the microcontroller and the wireless transceiver. This figure shows the same temperature threshold, but with a 3.3V output voltage.

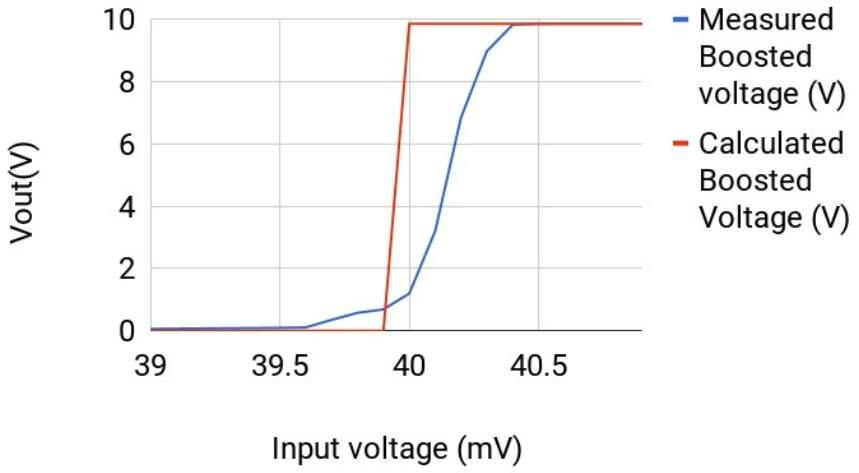


Figure 13. DC-DC Booster (VB-0410-2 Gold) calculated and measured output voltage as a function of its input.

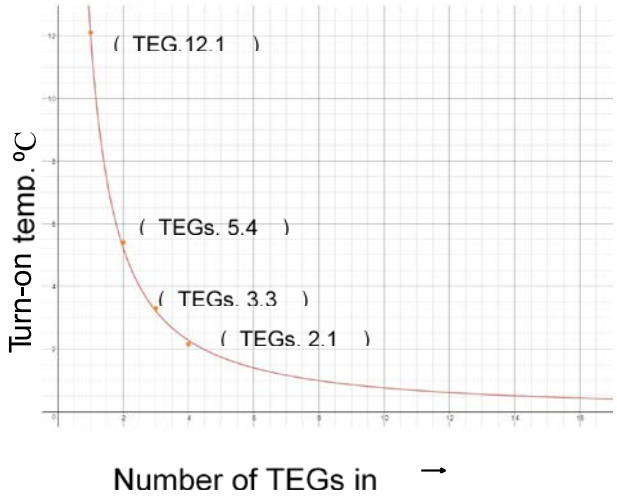


Figure 14. Estimated turn-on temperatures as a function of the number of TEG elements in a harvester unit.

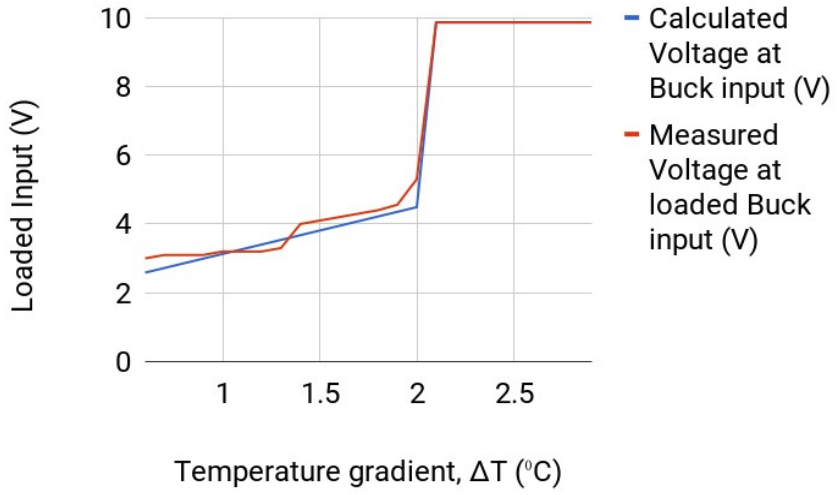


Figure 15. Calculated and measured voltage input to the loaded buck converter as a function of ΔT using a four-unit TEG harvester.

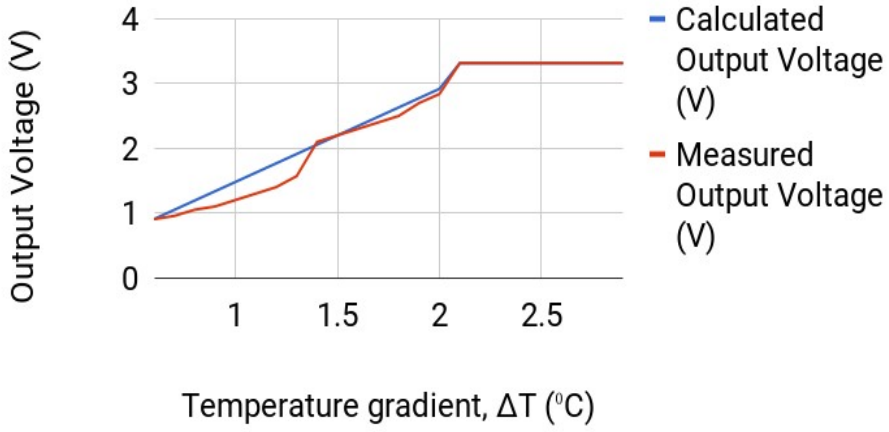


Figure 16. TI TPS54160 buck converter output voltage.

5.2.2. Estimating SHM System Downtime

Earlier measurements and plots show that the minimum required Δ*T* with a four-unit TEG harvester is approximately 2.1°C. The earlier analysis of the temperature profiles in asphalt concrete pavements allowed computing the number of hours per year when there will be a temperature differential lower than 2.1°C and hence, the system will be down for lack of power. As mentioned earlier, the length of time when the minimum 2.1°C temperature differential is available can be expanded by installing a 2-tier TEG harvester connected in parallel, one driven by the 0.02m-0.15m temperature differential and the other by the 0.02m-0.50m temperature differential. This 2-tier design allows power generation from at least one of the harvesters when

the temperature differential driving the other drops below the 2.1°C temperature differential threshold (Figure 5). These two TEG harvester units can be connected to the same booster as shown in Figure 17.

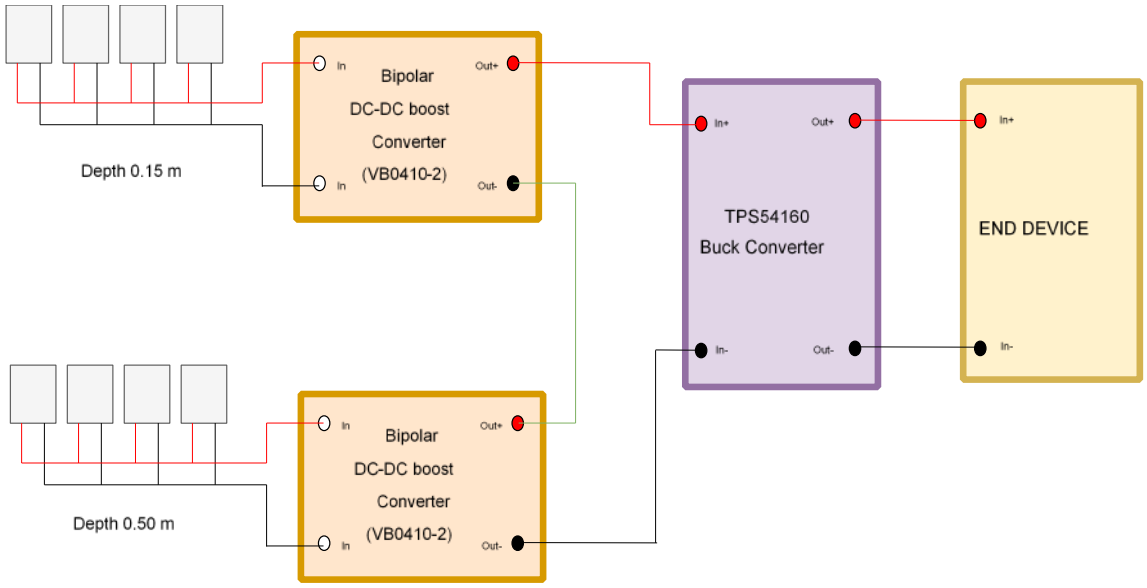


Figure 17. System block diagram for the 2-tier design TEG harvester.

Assuming that the SHM system uses a 2-tier TEG harvester design, its downtime can be computed from the length of time the temperature differential drops below 2.1°C simultaneously for both harvesters. The results of the thermal profile analysis presented earlier was used for this purpose. Analysis was conducted for both the South Texas and Northern Minnesota locations. Table 1 shows the number of hours the SHM system has insufficient power to operate as a function of the number of TEG units per harvester. Figure 18 plots the results in terms of % off period per year. It is noted that the two harvester prototypes developed had 2 or 4 TEG generators each. The differential cost between these two designs is marginal.

Table 1. Number of hours per year the SHM system is down.

No. of TEG units per Harvester	Annual off-time in Texas (hours)	Annual off-time in Minnesota (hours)
1	2815	2410
2	508	368
3	115	98
4	30	18

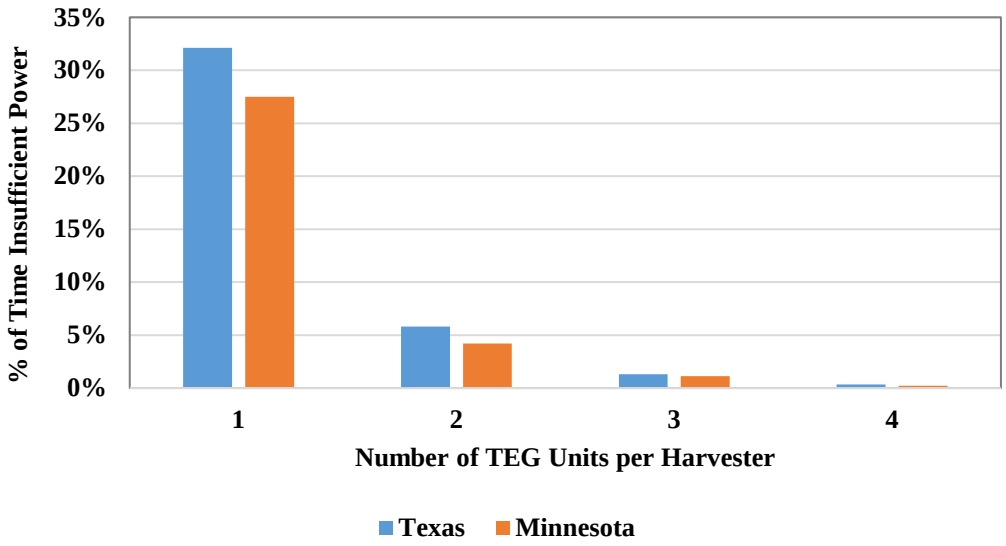


Figure 18. Percentage of time there is insufficient power to operate the SHM system (2-Tier TEG harvester).

5.3. SHM System Data Storage and Transmission Capabilities

The CC2500 transceiver (on the EZ4300-RF2500 chip) used in the prototype SHM system transmits data at a minimum rate of 1.2 kbps using approximately 3mW power. The MSP430F2274 microcontroller has a 10-bit ADC. The data storage requirement depends on the data being recorded. As an example, to record the peak load value for one of the axles measured by the *RoadTrax BL* sensor (Figure 10) would require 3 bytes (i.e., 24 bits). In this case, the internal data storage and transmission requirements would be a function of the volume and configuration of the traffic at a particular site (i.e., number of vehicles/day and percent of trucks by class). Regardless, if the quantity recorded was a single number, it would take a very long time to fill an inexpensive memory card. For example, a heavily trafficked lane carrying 10,000 cars per day will generate 60,000 bytes of data per day, which in turn will take over 700 years to fill a 16 GB memory card. Hence, data storage is not an issue for this type of application.

Data transmission however, has limitations. Data needs to be transmitted from the End Device to the Access Point of the system through the wireless transceiver CC2500. As mentioned earlier, the transceiver is either at roadside or is carried by a pilot car that is dedicated to collect data throughout a roadway network. Either way, powering the Access Point is not an issue. The power required by the End Device, whoever, is a function of the data transmission range and the data transmission rates. Table 2 shows the data transfer rates and the corresponding power consumption and range of the CC2500 transceiver.

Table 2. CC2500 data transfer rates, transmission range and power consumption.

Power (mW)	Range (m)	Transmission time (s)	Data transfer (kb)
3.18	18	0.81	0.966
24.2	44	1.97	98.43
34.5	50	2.24	223.7
41.1	55	2.46	369.1
48.7	65	2.91	727.1
60.3	72	3.22	1288.6
70	75	3.36	1677.8

A 2-tier harvester design with a four-unit TEG each will need to generate at least 24.2 mW to transmit 98.43 kb in 1.97 sec (i.e., rate of only 50 kbps) over a range of 44 m, so a pilot vehicle travelling at 50 mph (22.22 m/sec) can retrieve it without slowing down. Larger size data would require that the pilot vehicle slows down or even stops. Alternatively, the transmission rate/range could be increased posing additional SHM system powering requirements. Doing so is feasible by drastically increasing the number of TEG units per harvester. Even so, such a system may not be able to transmit at very high rates when there is insufficient temperature differential to power it. There are two potential solutions to this issue:

- Introduce a storage battery, and
- Transmit only when the Access Point is nearby, otherwise keeping the End Device in "sleep" mode which consumes only 1.3 mW. This would require a mechanism for "waking up" the End Device.

These two potential enhancements to the SHM system are feasible but were not implemented in this study, because they did not satisfy the system requirements set forth. They are under consideration in developing the next generation of the SHM system.

6. CONCLUSIONS

The SHM system developed meets the requirements set forth by this study. It is self-powered, battery-less and provides continuous data storage and wireless transmission. It is very versatile, accepting input from any sensor that outputs analog voltage. The user simply needs to program the transfer function into the microprocessor and have the system store the value of interest (e.g., peak axle load, peak strain and so on). The 2-tier TEG harvester provides sufficient energy to keep the system powered for temperature differentials as low as 2.1°C. Analyzing the temperature distributions in asphalt pavements showed that this is possible 95% of the time. Storing such data does not seem to be an issue, given the low cost of flush memory. Transmitting this data is an issue, since the End Device is the most power demanding function of the SHM system. The *SimpliciTI* protocol used allows relatively low transmission rates which make drive-by retrieval limited (i.e., only 50 kb of data can be read at 50 mph).

7. RECOMMENDATIONS

The highest power consumption component of the SHM system developed is the End Device wireless data transmission. Replacing the *SimpliciTI* protocol used for transmission with a lower power transceiver such one using the Bluetooth Low Energy (BLE) protocol could reduce power requirements, increase data transmission speed and expand transmission range. As pointed out earlier, another potential improvement would be to set the End Device to "sleep" mode while there is no Access Point in the vicinity. The energy saved from doing so could charge a storage battery that could power the SHM continuously, regardless of the temperature differentials applied to the TEG harvester. Needless to say, a mechanism for "waking up" the End Device would be needed and the battery would increase the maintenance requirements for the SHM system.

REFERENCES

1. Texas Instruments. A brief tutorial on SimpliciTI 1.1.1, November 2010. http://senstools.gforge.inria.fr/lib/exe/fetch.php?media=lib:simpliciti_brief_tutorial.pdf. Accessed June 11, 2018.

2. Texas Instruments Wiki. The eZ430-RF2500 Development Tool. Access point. http://processors.wiki.ti.com/index.php/EZ430-RF2500. Accessed June 11, 2018.

3. Wikimedia Commons. File: Buck_Circuit_Diagram.svg, DC buck converter. https://commons.wikimedia.org/w/index.php?curid=8687723. Accessed June 11, 2018.

4. Custom Thermoelectric. ELC-VB0410-2 Bipolar Voltage Booster 40mV. https://customthermoelectric.com/elc-vb0410-2-bipolar-voltage-booster-40mv.html. Accessed June 11, 2018.

5. Texas Instruments. TI Power Management Lab Kit (TI-PMLK), https://university.ti.com/en/faculty/teaching-materials-and-classroom-resources/ti-based-teaching-kits-for-analog-and-power-design/power-management-lab-kit-series. Accessed June 11, 2018.

6. Custom Thermoelectric. T.X.L-287-03.Z T.E.G. 62 x 62mm. https://customthermoelectric.com/txl-287-03z-teg-62-x-62mm.html. Accessed June 11, 2018.

7. Datta, U., Dessouky, S., and Papagiannakis, A.T. (2017) Harvesting of Thermoelectric Energy from Asphalt Pavements, TRB Paper 17-05481, Journal of the Transportation Research Board Record No. 2628, http://dx.doi.org/10.3141/2628-02

8. Gilbert, J.M. and Balouchi, F. (2008) Comparison of energy harvesting system for wireless sensor networks, International Journal of Automation and Computing 5(4):334-347, DOI: 10.1007/s11633-008-0334-2

9. Hassan, C., Man, S.-H., and Chang, C.-C. (2012) Development of energy harvested wireless sensing node for structural health monitoring, IEEE, pp. 22–27.

10. Long Term Pavement Performance Database On-Line, https://infopave.fhwa.dot.gov/ (2018).

11. Lynch, J.P. and Loh, K.J. (2006) A summary review of wireless sensors and sensor networks for structural health monitoring," The Shock and Vibration Digest, 1(2), pp. 3–12.

12. Saida, M., Zaibi, G., and Kachouri, A. (2017) A new design of thermoelectric generator for health monitoring, International Conference on Smart, Monitored and Controlled Cities (SM2C).

13. Tan, Y.K., and Panda, S.K. (2010) Review of Energy Harvesting Technologies for Sustainable Wireless Sensor Network, www.intechopen.com

14. Thermoelectric generator. https://en.wikipedia.org/wiki/Thermoelectric_generator. Accessed 23 May, 2016.

15. U. Nevada Reno, (2014), Temperature Estimate Model for Pavement Structures (TEMPS) www.arc.unr.edu/Software.html#TEMPS

16. Zhou, D., and Kong, N. (2010) A self-powered wireless sensor node for structural health monitoring, in SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring, pp. 2–13, SPIE.

APPENDIX A

A.1. End Device Code for Communicating with Access Point

In the prototype system, the Transceiver CC2500 is used together with the MSP430F2274 controller in the End Block, as well as in the Access Point. The Access Point collects data from the End Device. The custom code below was developed in this study for this purpose which can be uploaded to the receiver via the USB port.

```
#include "bsp.h"
#include "mrfi.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "vlo_rand.h"
//#include  "msp430g2253.h"
/*------------------------------------------------------------------------------
 * Defines
 *------------------------------------------------------------------------------*/
/* How many times to try a TX and miss an acknowledge
 before doing a scan */
#define MISSES_IN_A_ROW  2
/*------------------------------------------------------------------------------
 * Prototypes
 *------------------------------------------------------------------------------*/
static void linkTo(void);
void createRandomAddress(void);
__interrupt void ADC10_ISR(void);
__interrupt void Timer_A (void);
/*------------------------------------------------------------------------------
* Globals
------------------------------------------------------------------------------*/
static linkID_t sLinkID1 = 0;
/* Temperature offset set at production */
volatile int * tempOffset = (int *)0x10F4;
/* Initialize radio address location */
char * Flash_Addr = (char *)0x10F0;
/* Work loop semaphores */
static volatile uint8_t sSelfMeasureSem = 0;
// Global variables
int adc[10] = {0}; //Sets up an array of 10 integers
and zero's the values
int avg_adc = 0;
//MRFI_Init();
// Function prototypes
void adc_Setup();
void adc_Sam10();
```

```
/*-------------------------------------------------------------------------
 * Main
 *-------------------------------------------------------------------------*/
void main (void)
{
  addr_t lAddr;
  /* Initialize board-specific hardware */
  BSP_Init();
  //WDTCTL = WDTPW + WDTHOLD; // Stop WDT
  adc_Setup();
  /* Check flash for previously stored address */
  if(Flash_Addr[0] == 0xFF && Flash_Addr[1] == 0xFF &&
  Flash_Addr[2] == 0xFF && Flash_Addr[3] == 0xFF )
  {
    createRandomAddress(); // Create and store a new
    random address
  }
  /* Read out address from flash */
  lAddr.addr[0] = Flash_Addr[0];
  lAddr.addr[1] = Flash_Addr[1];
  lAddr.addr[2] = Flash_Addr[2];
  lAddr.addr[3] = Flash_Addr[3];
  /* Tell network stack the device address */
  SMPL_Ioctl(IOCTL_OBJ_ADDR, IOCTL_ACT_SET, &lAddr);
  /* Initialize TimerA and oscillator */
  BCSCTL3 |= LFXT1S_2;              // LFXT1 = VLO
  TACCTL0 = CCIE;             // TACCR0 interrupt enabled
  TACCR0 = 24000;             // ~ 1 sec
  TACTL = TASSEL_1 + MC_1;                  // ACLK, upmode
  while (SMPL_SUCCESS != SMPL_Init(0))
  {
    //BSP_TOGGLE_LED1();
    //BSP_TOGGLE_LED2();
    /* Go to sleep (LPM3 with interrupts enabled)
     * Timer A0 interrupt will wake CPU up every second
    to retry initializing
*/
    __bis_SR_register(LPM3_bits+GIE);
     // LPM3 with interrupts enabled
  }
  /* LEDs on solid to indicate successful join. */
  //BSP_TURN_ON_LED1();
  //BSP_TURN_ON_LED2();
  /* Unconditional link to AP which is listening due
  to successful join. */
  linkTo();
```

```
  while(1);
}
static void linkTo()
{
  uint8_t msg[3];
#ifdef APP_AUTO_ACK
  uint8_t misses, done;
#endif
  /* Keep trying to link... */
  while (SMPL_SUCCESS != SMPL_Link(&sLinkID1))
  {
    //BSP_TOGGLE_LED1();
    //BSP_TOGGLE_LED2();
    /* Go to sleep (LPM3 with interrupts enabled)
     * Timer A0 interrupt will wake CPU up every second
    to retry linking
*/
    __bis_SR_register(LPM3_bits+GIE);
  }
  /* Turn off LEDs. */
  //BSP_TURN_OFF_LED1();
  //BSP_TURN_OFF_LED2();
  /* Put the radio to sleep */
  SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, 0);
  while (1)
  {
    /* Go to sleep, waiting for interrupt every second
    to acquire data */
    __bis_SR_register(LPM3_bits);
      adc_Sam10();      // Function call for adc_samp
      // Add all the sampled data and divide by 10 to
      //find average
      avg_adc = ((adc[0]+adc[1]+adc[2]+adc[3]+adc[4]+
      adc[5]+adc[6]+adc[7]+adc[8]+adc[9]) / 10);
      //avg_adc = adc[0];
      msg[2]=(avg_adc/100);
      msg[1]=(avg_adc%100/10);
      msg[0]=(avg_adc%10);
      //msg[2] = 2;
      //msg[1] = 1;
      //msg[0] = 0;
      /* Get radio ready...awakens in idle state */
      SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_AWAKE, 0);
#ifdef APP_AUTO_ACK
      /* Request that the AP sends an ACK back to confirm
      data transmission
```

```
        * Note: Enabling this section more than DOUBLES the
        current consumption due to the amount of time IN RX
        waiting for
        the AP to respond
        */
       done = 0;
       while (!done)
       {
         noAck = 0;
         /* Try sending message MISSES_IN_A_ROW
         times looking for ack */
         for (misses=0; misses < MISSES_IN_A_ROW; ++misses)
         {
           if (SMPL_SUCCESS == (rc=SMPL_SendOpt(sLinkID1,
           msg, sizeof(msg), SMPL_TXOPTION_ACKREQ)))
           {
             /* Message acked. We're done. Toggle LED 1
             to indicate ack received. */
             BSP_TURN_ON_LED1();
             __delay_cycles(2000);
             BSP_TURN_OFF_LED1();
             break;
           }
           if (SMPL_NO_ACK == rc)
           {
             /* Count ack failures. Could also fail
             becuase of CCA and
              * we don't want to scan in this case.
              */
             noAck++;
           }
         }
         if (MISSES_IN_A_ROW == noAck)
         {
           /* Message not acked */
           BSP_TURN_ON_LED2();
           __delay_cycles(2000);
           BSP_TURN_OFF_LED2();
#ifdef FREQUENCY_AGILITY
           /* Assume we're on the wrong channel so
           look for channel by
            * using the Ping to initiate a scan when
            it gets no reply. With
            * a successful ping try sending the
            message again. Otherwise,
            * for any error we get we will wait
```

```
          until the next button
          * press to try again.
          */
         if (SMPL_SUCCESS != SMPL_Ping(sLinkID1))
         {
           done = 1;
         }
#else
        done = 1;
#endif  /* FREQUENCY_AGILITY */
       }
       else
       {
         /* Got the ack or we don't care. We're done. */
         done = 1;
       }
     }
#else
     /* No AP acknowledgement, just send a single
     message to the AP */
     SMPL_SendOpt(sLinkID1, msg, sizeof(msg), SMPL_TXOPTION_NONE);
#endif /* APP_AUTO_ACK */
     /* Put radio back to sleep */
     SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, 0);
     /* Done with measurement, disable measure flag */
     sSelfMeasureSem = 0;
   }
  }
void createRandomAddress()
{
  unsigned int rand, rand2;
  do
  {
    rand = TI_getRandomIntegerFromVLO();
    // first byte can not be 0x00 of 0xFF
  }
  while( (rand & 0xFF00)==0xFF00 || (rand & 0xFF00)==0x0000 );
  rand2 = TI_getRandomIntegerFromVLO();
  BCSCTL1 = CALBC1_1MHZ;
  // Set DCO to 1MHz
  DCOCTL = CALDCO_1MHZ;
  FCTL2 = FWKEY + FSSEL0 + FN1;
  // MCLK/3 for Flash Timing Generator
  FCTL3 = FWKEY + LOCKA;
  // Clear LOCK & LOCKA bits
  FCTL1 = FWKEY + WRT;
```

```
  // Set WRT bit for write operation
  Flash_Addr[0]=(rand>>8) & 0xFF;
  Flash_Addr[1]=rand & 0xFF;
  Flash_Addr[2]=(rand2>>8) & 0xFF;
  Flash_Addr[3]=rand2 & 0xFF;
  FCTL1 = FWKEY;
  // Clear WRT bit
  FCTL3 = FWKEY + LOCKA + LOCK;
  // Set LOCK & LOCKA bit
}
/*-------------------------------------------------------------------------
 * ADC10 interrupt service routine
 *------------------------------------------------------------------------*/
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);
  // Clear CPUOFF bit from 0(SR)
}
/*-------------------------------------------------------------------------
 * Timer A0 interrupt service routine
 *------------------------------------------------------------------------*/
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  sSelfMeasureSem = 1;
  __bic_SR_register_on_exit(LPM3_bits);
  // Clear LPM3 bit from 0(SR)
}
// ADC10 interrupt service routine
// ADC set-up function
void adc_Setup()
{
    ADC10CTL1 = CONSEQ_2 + INCH_0;
    // Repeat single channel, A0
    ADC10CTL0 = ADC10SHT_2 + MSC + ADC10ON + ADC10IE;
    // Sample & Hold Time + ADC10 ON + Interrupt Enable
    ADC10DTC1 = 0x01;
    // 10 conversions
    ADC10AE0 |= 0x0A;
    // P1.0 ADC option select
}
// ADC sample conversion function
void adc_Sam10()
{
    ADC10CTL0 &= ~ENC;
```

```
    // Disable Conversion
    while (ADC10CTL1 & BUSY);
    // Wait if ADC10 busy
    ADC10SA = (int)adc;
    // Transfers data to next array (DTC auto increments address)
    ADC10CTL0 |= ENC + ADC10SC;
    // Enable Conversion and conversion start
    __bis_SR_register(CPUOFF + GIE);
    // Low Power Mode 0, ADC10_ISR
}
```

A.2. Access Point Code

```
#include <string.h>
#include <stdio.h>
#include "bsp.h"
#include "mrfi.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "nwk_frame.h"
#include "nwk.h"
#include "virtual_com_cmds.h"
/* Frequency Agility helper functions */
static void    checkChangeChannel(void);
static void    changeChannel(void);
__interrupt void ADC10_ISR(void);
__interrupt void Timer_A (void);
/*------------------------------------------------------------------------------
 * Globals
 *------------------------------------------------------------------------------*/
/* reserve space for the maximum possible peer Link IDs */
static linkID_t sLID[NUM_CONNECTIONS] = {0};
static uint8_t  sNumCurrentPeers = 0;
/* callback handler */
static uint8_t sCB(linkID_t);
/* received message handler */
static void processMessage(linkID_t, uint8_t *, uint8_t);
/* work loop semaphores */
static volatile uint8_t sPeerFrameSem = 0;
static volatile uint8_t sJoinSem = 0;
static volatile uint8_t sSelfMeasureSem = 0;
/* blink LEDs when channel changes... */
static volatile uint8_t sBlinky = 0;
/* data for terminal output */
const char splash[] = {"\r\n---------
----------------------------------------  \r\n     ****\r\n
****           eZ430-RF2500\r\n     ******o****
Temperature Sensor Network\r\n********_///_****
Copyright 2009\r\n ******/_//_/*****  Texas Instruments
Incorporated\r\n  ** ***(__/*****   All rights reserved.\r\n
*********     SimpliciTI1.1.1\r\n       *****\r\n       ***\r\n---------------------------------------
----------\r\n"};
volatile int * tempOffset = (int *)0x10F4;
/*------------------------------------------------------------------------------
 * Frequency Agility support (interference detection)
 *------------------------------------------------------------------------------*/
```

```
#ifdef FREQUENCY_AGILITY

#define INTERFERNCE_THRESHOLD_DBM (-70)
#define SSIZE    25
#define IN_A_ROW  3
static int8_t  sSample[SSIZE];
static uint8_t sChannel = 0;
#endif  /* FREQUENCY_AGILITY */
char printing[] = {"\r\nXXX"};
int value=0;
/*-------------------------------------------------------------------------
 * Main
 *-------------------------------------------------------------------------*/
void main (void)
{
  bspIState_t intState;
#ifdef FREQUENCY_AGILITY
  memset(sSample, 0x0, sizeof(sSample));
#endif
  /* Initialize board */
  BSP_Init();

  /* Initialize TimerA and oscillator */
  BCSCTL3 |= LFXT1S_2;
  // LFXT1 = VLO
  TACCTL0 = CCIE;
  // TACCR0 interrupt enabled
  TACCR0 = 12000;
  // ~1 second
  TACTL = TASSEL_1 + MC_1;
  // ACLK, upmode
  /* Initialize serial port */
  COM_Init();
  //sprintf(printing,"Started");
  //Transmit splash screen and network init notification
  TXString( (char*)splash, sizeof splash);
  TXString( "\r\nInitializing Network....", 26 );
  SMPL_Init(sCB);
  // network initialized
  TXString( "Done\r\n", 6);
  /* green and red LEDs on solid to indicate waiting
  for a Join. */
  BSP_TURN_ON_LED1();
  BSP_TURN_ON_LED2();
  /* main work loop */
  while (1)
```

```
{
 /* Wait for the Join semaphore to be set by the
 receipt of a Join frame from
  * a device that supports an End Device.
  *
  * An external method could be used as well. A button
  press could be connected
  * to an ISR and the ISR could set a semaphore that
  is checked by a function
  * call here, or a command shell running in support
  of a serial connection
  * could set a semaphore that is checked by a function
  call.
  */
 if (sJoinSem && (sNumCurrentPeers < NUM_CONNECTIONS))
 {
   /* listen for a new connection */
   while (1)
   {
     if (SMPL_SUCCESS == SMPL_LinkListen
     (&sLID[sNumCurrentPeers]))
     {
       break;
     }
     /* Implement fail-to-link policy here.
     otherwise, listen again. */
   }
   sNumCurrentPeers++;
   BSP_ENTER_CRITICAL_SECTION(intState);
   sJoinSem--;
   BSP_EXIT_CRITICAL_SECTION(intState);
 }
 // if it is time to measure our own temperature...
 //if(sSelfMeasureSem)
 //{
   //char msg [6];
   //char addr[] = {"HUB0"};
   //char rssi[] = {"000"};
   //int degC, volt;
   //volatile long temp;
   //int results[2];
   /* Get temperature */
   //ADC10CTL1 = INCH_10 + ADC10DIV_4;
   // Temp Sensor ADC10CLK/5
   //ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON
   + ADC10ON + ADC10IE + ADC10SR;
```

```
    /* Allow ref voltage to settle for at least
    30us (30us * 8MHz = 240 cycles)
     * See SLAS504D for settling time spec
     */
    //__delay_cycles(240);
    //ADC10CTL0 |= ENC + ADC10SC;
    // Sampling and conversion start
    //__bis_SR_register(CPUOFF + GIE);
    // LPM0 with interrupts enabled
    //results[0] = ADC10MEM;
    // Retrieve result
    //ADC10CTL0 &= ~ENC;
    /* Get voltage */
    //ADC10CTL1 = INCH_11;
    // AVcc/2
    //ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE +
REF2_5V;
    //__delay_cycles(240);
    //ADC10CTL0 |= ENC + ADC10SC;
    // Sampling and conversion start
    //__bis_SR_register(CPUOFF + GIE);
    // LPM0 with interrupts enabled
    //results[1] = ADC10MEM;
    // Retrieve result
    /* Stop and turn off ADC */
    //ADC10CTL0 &= ~ENC;
    //ADC10CTL0 &= ~(REFON + ADC10ON);
    /* oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV= A10*423/1024 - 278
     * the temperature is transmitted as an integer where 32.1 = 321
     * hence 4230 instead of 423
     */
    //temp = results[0];
    //degC = ((temp - 673) * 4230) / 1024;
    //if( (*tempOffset) != 0xFFFF )
    //{
     // degC += (*tempOffset);
    //}
    //temp = results[1];
    //volt = (temp*25)/512;
    /* Package up the data */
    //msg[0] = degC&0xFF;
    //msg[1] = (degC>>8)&0xFF;
    //msg[2] = volt;
    /* Send it over serial port */
    //transmitDataString(1, addr, rssi, msg );
    BSP_TOGGLE_LED1();
```

```
  /* Done with measurement, disable measure flag */
  sSelfMeasureSem = 0;
/* Have we received a frame on one of the ED connections?
 * No critical section -- it doesn't really matter much
 if we miss a poll
 */
if (sPeerFrameSem)
{
  uint8_t     msg[3], len, i;
  /* process all frames waiting */
  for (i=0; i<sNumCurrentPeers; ++i)
  {
    if (SMPL_SUCCESS == SMPL_Receive(sLID[i], msg, &len))
    {
      ioctlRadioSiginfo_t sigInfo;
      processMessage(sLID[i], msg, len);
      sigInfo.lid = sLID[i];
      SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SIGINFO,
      (void *)&sigInfo);

      value = msg[2]*100+msg[1]*10+msg[0];
      //transmitData( i, sigInfo.sigInfo.rssi,
      (char*)msg );
      printing[4]='0' + msg[0];
      printing[3]='0' + msg[1];
      printing[2]='0' + msg[2];
      TXString(printing, sizeof printing );
      BSP_TOGGLE_LED2();
      BSP_ENTER_CRITICAL_SECTION(intState);
      sPeerFrameSem--;
      BSP_EXIT_CRITICAL_SECTION(intState);
    }
  }
}
if (BSP_BUTTON1())
{
  __delay_cycles(2000000);  /* debounce (0.25 seconds) */
  changeChannel();
}
else
{
  checkChangeChannel();
}
BSP_ENTER_CRITICAL_SECTION(intState);
if (sBlinky)
{
```

```
      if (++sBlinky >= 0xF)
      {
        sBlinky = 1;
        BSP_TOGGLE_LED1();
        BSP_TOGGLE_LED2();
      }
    }
    BSP_EXIT_CRITICAL_SECTION(intState);
  }
}
/* Runs in ISR context. Reading the frame should be done
in the */
/* application thread not in the ISR thread. */
static uint8_t sCB(linkID_t lid)
{
  if (lid)
  {
    sPeerFrameSem++;
    sBlinky = 0;
  }
  else
  {
    sJoinSem++;
  }
  /* leave frame to be read by application. */
  return 0;
}
static void processMessage(linkID_t lid,
uint8_t *msg, uint8_t len)
{
  /* do something useful */
  if (len)
  {
    //sprintf(printing, "a %d",*msg);
    //sprintf(printing, "Second element %d",msg[1]);
      //transmitData( 0, sigInfo.sigInfo.rssi, (char*)msg );
    BSP_TOGGLE_LED1();
  }
  return;
}
static void changeChannel(void)
{
#ifdef FREQUENCY_AGILITY
  freqEntry_t freq;
  if (++sChannel >= NWK_FREQ_TBL_SIZE)
  {
```

```
    sChannel = 0;
  }
  freq.logicalChan = sChannel;
  SMPL_Ioctl(IOCTL_OBJ_FREQ, IOCTL_ACT_SET, &freq);
  BSP_TURN_OFF_LED1();
  BSP_TURN_OFF_LED2();
  sBlinky = 1;
#endif
  return;
}
/* implement auto-channel-change policy here... */
static void checkChangeChannel(void)
{
#ifdef FREQUENCY_AGILITY
  int8_t dbm, inARow = 0;
  uint8_t i;
  memset(sSample, 0x0, SSIZE);
  for (i=0; i<SSIZE; ++i)
  {
    /* quit if we need to service an app frame */
    if (sPeerFrameSem || sJoinSem)
    {
      return;
    }
    NWK_DELAY(1);
    SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RSSI,
    (void *)&dbm);
    sSample[i] = dbm;
    if (dbm > INTERFERNCE_THRESHOLD_DBM)
    {
      if (++inARow == IN_A_ROW)
      {
        changeChannel();
        break;
      }
    }
    else
    {
      inARow = 0;
    }
  }
#endif
  return;
}
/*---------------------------------------------------------------------------
* ADC10 interrupt service routine
```

```
-------------------------------------------------------------------------------*/
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);
  // Clear CPUOFF bit from 0(SR)
}

/*-------------------------------------------------------------------------------
* Timer A0 interrupt service routine
-------------------------------------------------------------------------------*/
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  sSelfMeasureSem = 0;
}
```

A.3. Minimal RF Integration Code

```
#ifndef MRFI_H
#define MRFI_H


#include "bsp.h"
#include "mrfi_defs.h"
/* ------------------------------------------------------------------------------------------
 *                                      Defines
 * ------------------------------------------------------------------------------------------
 */
#define MRFI_NUM_LOGICAL_CHANS
__mrfi_NUM_LOGICAL_CHANS__
#define MRFI_NUM_POWER_SETTINGS
__mrfi_NUM_POWER_SETTINGS__
/* return values for MRFI_Transmit */
#define MRFI_TX_RESULT_SUCCESS       0
#define MRFI_TX_RESULT_FAILED        1
/* transmit type parameter for MRFI_Transmit */
#define MRFI_TX_TYPE_FORCED          0
#define MRFI_TX_TYPE_CCA             1
#ifndef SMPL_SECURE
#define  NWK_HDR_SIZE   3
#define  NWK_PAYLOAD    MAX_NWK_PAYLOAD
#else
#define  NWK_HDR_SIZE   6
#define  NWK_PAYLOAD    (MAX_NWK_PAYLOAD+4)
#endif
/* if external code has defined a maximum payload,
use that instead of default */
#ifdef MAX_APP_PAYLOAD
#ifndef MAX_NWK_PAYLOAD
#error ERROR: MAX_NWK_PAYLOAD not defined
#endif
#if MAX_APP_PAYLOAD < NWK_PAYLOAD
#define MAX_PAYLOAD  NWK_PAYLOAD
#else
#define MAX_PAYLOAD  MAX_APP_PAYLOAD
#endif
#define MRFI_MAX_PAYLOAD_SIZE  (MAX_PAYLOAD+NWK_HDR_SIZE)
/* SimpliciTI payload size plus six byte overhead */
#endif
/* frame definitions */
#define MRFI_ADDR_SIZE             __mrfi_ADDR_SIZE__
```

```
#ifndef MRFI_MAX_PAYLOAD_SIZE
#define MRFI_MAX_PAYLOAD_SIZE       __mrfi_MAX_PAYLOAD_SIZE__
#endif
#define MRFI_MAX_FRAME_SIZE
(MRFI_MAX_PAYLOAD_SIZE + __mrfi_FRAME_OVERHEAD_SIZE__)
#define MRFI_RX_METRICS_SIZE
__mrfi_RX_METRICS_SIZE__
#define MRFI_RX_METRICS_RSSI_OFS
__mrfi_RX_METRICS_RSSI_OFS__
#define MRFI_RX_METRICS_CRC_LQI_OFS
__mrfi_RX_METRICS_CRC_LQI_OFS__
/* Radio States */
#define MRFI_RADIO_STATE_UNKNOWN  0
#define MRFI_RADIO_STATE_OFF      1
#define MRFI_RADIO_STATE_IDLE     2
#define MRFI_RADIO_STATE_RX       3
/* Platform constant used to calculate worst-case for
an application
 * acknowledgment delay. Used in the NWK_REPLY_DELAY().
 *
#define   PLATFORM_FACTOR_CONSTANT
(2 +
2*(MAX_HOPS*(MRFI_CCA_RETRIES*(8*MRFI_BACKOFF_PERIOD_USECS)/1000)
))
/* ------------------------------------------------------------------------------------------
 *      Macros
 * ------------------------------------------------------------------------------------------
 */
#define MRFI_GET_PAYLOAD_LEN(p)
__mrfi_GET_PAYLOAD_LEN__(p)
#define MRFI_SET_PAYLOAD_LEN(p,x)
__mrfi_SET_PAYLOAD_LEN__(p,x)

#define MRFI_P_DST_ADDR(p)
__mrfi_P_DST_ADDR__(p)
#define MRFI_P_SRC_ADDR(p)
__mrfi_P_SRC_ADDR__(p)
#define MRFI_P_PAYLOAD(p)
__mrfi_P_PAYLOAD__(p)

/* ------------------------------------------------------------------------------------------
 *       Typdefs
 * ------------------------------------------------------------------------------------------
 */
```

```
typedef struct
{
  uint8_t frame[MRFI_MAX_FRAME_SIZE];
  uint8_t rxMetrics[MRFI_RX_METRICS_SIZE];
} mrfiPacket_t;
/* ------------------------------------------------------------------------------------------
 *       Prototypes
 * ------------------------------------------------------------------------------------------
 */
void    MRFI_Init(void);
uint8_t MRFI_Transmit(mrfiPacket_t *, uint8_t);
void    MRFI_Receive(mrfiPacket_t *);
void    MRFI_RxCompleteISR(void);
/* populated by code using MRFI */
uint8_t MRFI_GetRadioState(void);
void    MRFI_RxOn(void);
void    MRFI_RxIdle(void);
int8_t  MRFI_Rssi(void);
void    MRFI_SetLogicalChannel(uint8_t);
uint8_t MRFI_SetRxAddrFilter(uint8_t *);
void    MRFI_EnableRxAddrFilter(void);
void    MRFI_DisableRxAddrFilter(void);
void    MRFI_Sleep(void);
void    MRFI_WakeUp(void);
uint8_t MRFI_RandomByte(void);
void    MRFI_DelayMs(uint16_t);
void    MRFI_ReplyDelay(void);
void    MRFI_PostKillSem(void);
void    MRFI_SetRFPwr(uint8_t);
/* ------------------------------------------------------------------------------------------
 *    Global Constants
 * ------------------------------------------------------------------------------------------
 */
extern const uint8_t mrfiBroadcastAddr[];
#endif
```