UMR
UNIVERSITY OF MISSOURI-ROLLA

CENTER FOR INFRASTRUCTURE ENGINEERING STUDIES

The Flood Frog: an Autonomous Wireless Device for Flood Detection and Monitoring

By

Valerio Plessi

Filippo Bastianini

Sahra Sedighsarvestani

University Transportation Center Program
at the University of Missouri-Rolla

Technical Report Documentation Page

1. Report No. UTC R158	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle The Flood Frog: an Autonomous Wireless Device for Flood Detection and Monitoring		5. Report Date January 2007
		6. Performing Organization Code
7. Author/s Valerio Plessi, Filippo Bastianini, Sahra Sedighsarvestani		8. Performing Organization Report No. 00001406/0008404
9. Performing Organization Name and Address Center for Infrastructure Engineering Studies/UTC program University of Missouri - Rolla 223 Engineering Research Lab Rolla, MO 65409		10. Work Unit No. (TRAIS)
		11. Contract or Grant No. DTRS98-G-0021
12. Sponsoring Organization Name and Address U.S. Department of Transportation Research and Special Programs Administration 400 7th Street, SW Washington, DC 20590-0001		13. Type of Report and Period Covered Final
		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract
This report describes the real-time data acquisition, communication, and alerting capabilities of the Flood Frog, an embedded an autonomous device equipped with multiple sensors for environmental and structural monitoring. The system is capable of operating for several years without human intervention. Battery power and utilization of the GSM mobile network result in a completely wireless system. Coupled with the low cost of the device, this allows deployment in locations where automatic monitoring was previously hindered by cost or infeasibility of installation.

17. Key Words embedded systems, environmental monitoring, real time information, wireless communication systems	18. Distribution Statement No restrictions. This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161.		
19. Security Classification (of this report) unclassified	20. Security Classification (of this page) unclassified	21. No. Of Pages 16	22. Price

Form DOT F 1700.7 (8-72)

The Flood Frog: an Autonomous Wireless Device for Flood Detection and Monitoring

Valerio Plessi, Filippo Bastianini and Sahra Sedighsarvestani
Center for Infrastructure Engineering Studies
University of Missouri-Rolla
Rolla, MO 65409-0040
{vp427, fbroptic, sedighs}@umr.edu

1 Introduction

Accurate real time monitoring is gaining critical importance in a broad range of domains. Of particular interest to this research is the development of monitoring and detection systems for environmental and structural phenomena. Transportation infrastructures, in particular bridges, provide examples for both categories. Flooding of low water bridges is one example of rapidly evolving environmental phenomena, while stress and vibration are structural phenomena in need of monitoring.

Early warning and advanced preparation for emergency are two of the most effective lines of defense against disasters such as floods, earthquakes, wildfire and hurricanes. The impact of disastrous events, including the recent hurricanes Katrina and Rita, underscores the limits of established early warning systems, especially with regard to rapidly evolving situations. Environmental monitoring, which refers to measuring and recording parameters such as temperature, humidity, acoustic emission and pollution in a selected site, enables early detection of catastrophic events. Ongoing provision of information facilitates recovery efforts and aids in containment of aftereffects.

Structural monitoring is another important issue, as periodic collection of information about the health of a structure, such as a bridge or a building, can prevent sudden breakdown, save money and most importantly, protect human lives. In this context, changes in stress and vibration can serve as warnings for impending structural damage. Regardless of the phenomenon being monitored, the information should be collected with resolution and frequency sufficient to enable accurate and timely evaluation of impending danger.

In monitoring applications, one major challenge is the infeasibility of installing the necessary devices in remote areas. As an example, low water bridges, which are prone to flooding, are typically located in rural areas that lack accessible power lines. The problem is further exacerbated by the costs associated with digging trenches and drawing the wires needed for a wired system, which can be prohibitively expensive.

Traditional structural health monitoring systems are based on a central unit that gathers information from several sampling nodes scattered throughout the site; each node is connected to others by wires that can be electrical or optical, depending on the type of equipment. In both cases, the installation and maintenance costs for the system are very high.

The delivery of information collected by wired systems is also challenging. Data transfer requires either a communication line to the external world, or a periodic inspections of the site. A related challenge arises from the high power needs of such instrumentation systems, which necessitates large batteries, or additional expenses associated with drawing traditional power lines. For large scale utilization to be feasible, monitoring costs should not exceed a small fraction (O(0.01)) of the overall construction costs. In [7], it is reported that the cost for a wired bridge monitoring system for 60 sampling points is approximately $300,000. This cost could be significantly reduced by eliminating power and communication lines.

A prevalent alternative to expensive wired structural monitoring systems is regular site visits by experts. This option is also expensive, especially for remote sites, but more importantly it delivers inconsistent results that are strongly biased by the competence of the individual performing the monitoring.

Environmental monitoring has also proven to be a costly proposition. In [1], the US Department of Energy estimates a cost of $24,000 for analyzing 4 acres of soil with traditional methods that require the drilling of boreholes and installation of monitoring wells. On a larger scale, the application of these techniques would be prohibitively expensive.

2 Approach

The challenges described in Section 1 underscore the necessity of a novel monitoring system that is less costly, more dependable and more flexible in terms of locations where it can be installed. Our proposed solution to the aforementioned challenges is an autonomous embedded system, termed the *Flood Frog*. The system is based upon a wireless network composed of inexpensive sensor nodes. Sensor networks have been successfully used in a broad range of applications, including habitat monitoring [11] to shooter localization [8]. The size, unobtrusiveness, and expendability of wireless sensor nodes make them ideal candidates for detection and monitoring systems.

In the approach proposed in this work, one or more sensor nodes are distributed in areas of interest. Each node is equipped with sensors that measure relevant environmental and structural parameters. These sensors collect data that is transmitted to a base station for processing. The base station, which serves as the core of the system, aggregates and processes the data received from the network, and when necessary, communicates to the external world the detection of an event of interest. The number of nodes and the particular sensors deployed will vary from one application to another, but the vast majority of applications share a number of basic requirements, in particular independence from power and transmission wires.

The salient features of the proposed approach are low cost, low power consumption, and wireless communication. The savings achieved by significantly decreasing installation and maintenance costs facilitates large scale deployment of the system for monitoring civil infrastructures or natural sites. Further savings and greater flexibility of usage are achieved due to the decrease in power consumption, which allows the use of small battery packs instead of large cells or traditional power lines. As an additional result, the system is completely autonomous since it does not need any cable.

The wireless nature of the network eliminates the considerable cost of drawing cables to the site, simplifies the design, and increases the robustness of the resulting system. In many cases, connecting a sensor to a wired network could be impossible, or at best, highly impractical. For accurate detection, the sensors often need to be mounted at very specific locations, for example, underneath a bridge, or at a high altitude, either of which would make the use of a wired network very inconvenient.

The system utilizes two wireless networks. The first is for short-range communication among the sensor nodes and between them and the base station. This network could be created on an ad hoc basis, which facilitates the incorporation of additional sensors and improves scalability of the system. The Zigbee protocol is an example of a suitable solution for this component of the system. The second network required by the proposed monitoring system is for long-range communication between the base station and the external world, to enable autonomous reporting of the collected data. Existing communication infrastructure, specifically the GSM mobile network, has been used to send the information by SMS, e-mail or FTP transactions. This choice ensures worldwide compatibility, as GSM is virtually ubiquitous.

The Flood Frog is an autonomous low cost and low power wireless device for monitoring. The design is general, and includes an A/D and a digital signal processing unit for local processing of data, which allows the use of any type of digital or analog sensor. The device can run for years on a standard battery pack, without requiring any human intervention. The operations are completely unattended and also the internal firmware is remotely updatable.

The design takes into consideration the hostile environment where the system is supposed to work. A waterproof box completely encloses the system, protecting the electronic components from outdoor elements such as water, humidity and dust. Thermal insulation is provided to keep the internal temperature as constant as possible to optimize the battery life. A tamper alarm is also supplied onboard to prevent vandalism or wildlife interference.

The structural and environmental phenomena are generally slow evolving. Quantities such as temperature, humidity, water level, tilt, displacement and stress have slow variations; therefore, a sampling period on the order of seconds or minutes will suffice. Flash flood is an example of a critical situation that the Flood Frog is able to monitor. According to the National Oceanic & Atmospheric Administration (NOAA) [10], flash floods can occur within a few minutes of excessive rainfall, a dam or levee failure or a sudden release of water held by an ice jam. For this phenomenon, a sensor sampling period of minutes can easily provide for real-time monitoring, with prompt alarm generation in case a threshold is exceeded.

According to the European Space Agency (ESA), flooding is the world's most costly type of natural disaster, as it can cause numerous human casualties and destroy housing, agriculture, and communication infrastructure [4]. In the United States alone, flooding has caused between 2 and 8 billion dollars of damage per year in the past decade [9].

Monitoring of flash floods is an application particularly suited to the capabilities of the Flood Frog. For this application, the base station alone suffices, as it contains all of the necessary hardware and software. To ensure the survivability of the device, an anti-tampering system is provided to prevent vandalism; moreover, a heartbeat mechanism is included to periodically communicate the status of the system, ensuring that it is still operational after a long unattended period.

3 Methodology

As the first field application of the device is flood monitoring, the hostile environment where the Flood Frog will be deployed should be taken into account in the design. For flood monitoring, the device will be installed near a riverbed or on a bridge girder, where submersion and high humidity are likely; hence, a water tight enclosure is necessary and has been provided. The box chosen is 7.5 x 5 x 4 inches, conforms to the IP 68 [3] standard, and protects against the effects of constant submersion in water. The small size of the device facilitates its installation in unattended places, where a conspicuous device would be subject to vandalism.

The box is completely sealed, with the exception of a small perforation for the water level probe. The entire system is enclosed in the case and has to operate wirelessly.

For the initial field application, the first problem to solve is the flood sensor. When the water reaches the threshold level, an alarm should be triggered. The easiest way to implement this is a floater inside a pipe, where the floater's position indicates the water level. In order to communicate this information to the device, a magnet is embedded inside the floater, while magnetic switches are installed inside the case. They have been placed at the border of the circuit board to be reachable by the magnetic field. The case design is depicted in Figure 1.

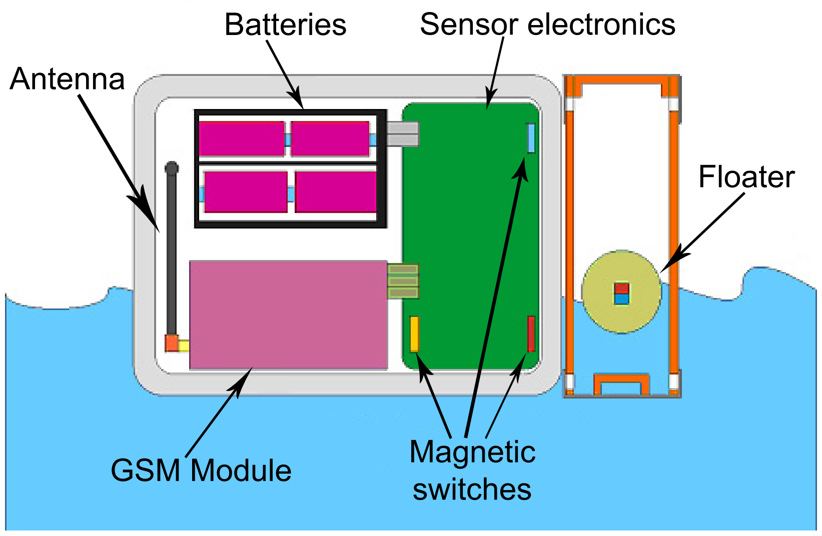

Figure 1. Case design of the Flood Frog.

On the right border of the sensor electronics are two magnetic switches. The lower switch (red in Figure 1) is closed when the water level is normal (no flooding), and checks the presence of the floater. The upper switch (light blue in Figure 1) triggers the alarm when the water reaches the set threshold level.

A third magnetic switch (yellow in Figure 1) is provided as an anti-tampering mechanism. It protects the device from vandalism and theft and is enclosed in a secure metal enclosure. On the enclosure, a magnet is placed very close to the yellow switch, and causes the switch to close when the case of the device is opened. This indicates tampering, and triggers an alarm.

The left side of Figure 2 depicts the IP 68 case with a test floater, and a four D-cell battery holder that is used as the sole power supply. The right side depicts the main board. As an added feature, solar panels can be added to the device to supplement the battery pack. This will necessitate a transparent box cover, which complicates waterproofing. The device can operate for several years with this standard battery pack, therefore the solar panels are more appropriate for use in smaller sensor nodes.

Figure 3 illustrates the hardware architecture of the Flood Frog. The power supply, located in the upper left corner of the figure, powers the entire device, and is connected to every component. For clarity, the wires connecting the power supply have not been depicted in the figure. The core of the device is comprised of two main components, the microcontroller unit (MCU) and the GSM module. They are connected through a serial interface, which can also be used to communicate with a PC for programming and debugging purposes. The RS232 block is a level translator necessary for communication between the PC serial port (DB-9 connector) and the MCU/GSM serial port. As the serial RS232 protocol provides only point-to-

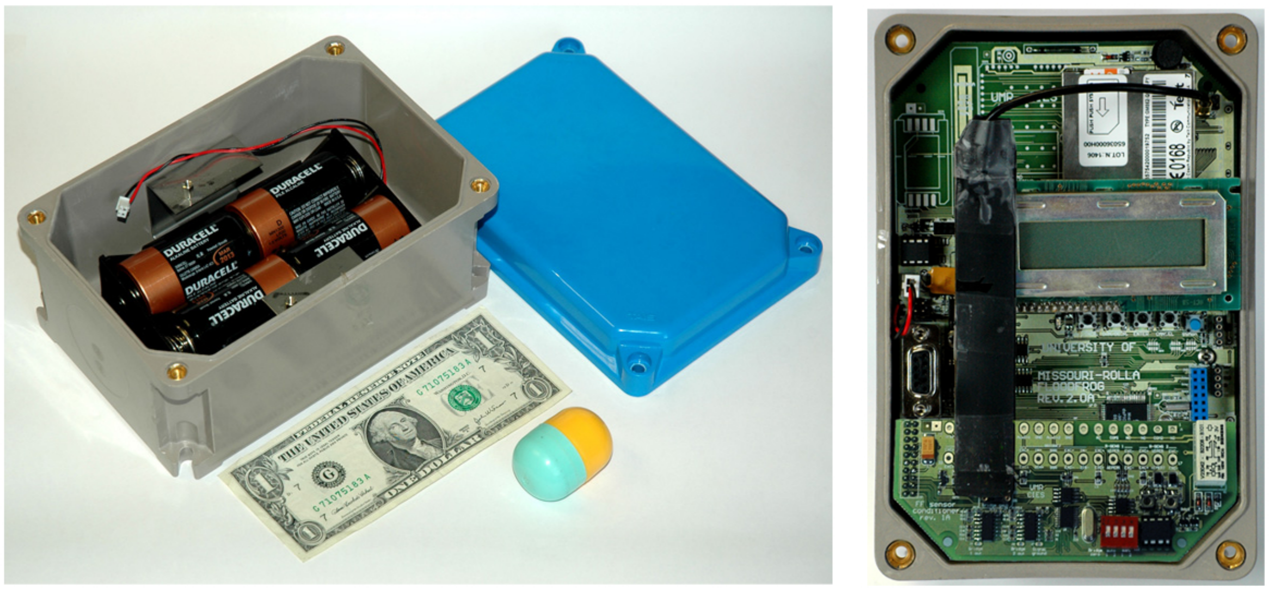

Figure 2. IP 68 box, batteries, magnetic floater and main board.

point communication, and does not support a bus, only one physical connection at a time (between two devices) is allowed, necessitating channel switching. The MCU and the GSM are also connected through additional wires that are control lines.

The GSM module is a quad-band mobile modem/phone and supports voice calls utilizing the microphone and the speaker. The module can also take pictures with the embedded camera, to be attached to multimedia messages or via GPRS. The MCU and the GSM module are connected by a Serial Peripheral Interface bus (SPI), which is a channel that allows bus-based serial communication. The bus also connects to external memory for data storage, and a wireless socket that can connect the Flood Frog to a local wireless sensor network. If a wireless sensor network is implemented, the Flood Frog will serve as the base station that supports several smaller sensor nodes.

The embedded sensors are directly mounted onto the main board and trigger alarms in case of flooding and/or enclosure tampering. The switches are connected to both the MCU and the GSM module. In fact, the GSM module is able to operate autonomously using its internal controller (i.e., without the MCU); depending on the application, the Flood Frog allows different operative configurations with one or both computational units working at the same time.

A temperature sensor sampled by the internal ADC of the MCU provides information about the temperature inside the device. The MCU also monitors the battery level. There are three additional sensor blocks; two of them are general purpose (boards 1 and 2) while the third is specifically designed for earthquake monitoring (i.e., vibration detection). The two general purpose boards are also linked to the SPI bus to allow direct communication with all the other connected peripherals.

The Micro Electro-Mechanical Systems (MEMS) sensor block embeds a 3-axis accelerometer that can be used to sense and measure motion, as well as tilt. The ADC of MCU digitizes the analog signals transmitted by all three sensor boards so they can be processed, stored and transmitted to the external world. For applications requiring a greater resolution than what is provided by the MCU, a higher resolution ADC is provided in the daughter board.

The MCU includes an interface to the In Circuit Debugger (ICD2©) integrated development environment. The ICD2 is a real time onboard programmer and debugger that facilitates the software development phase. To allow manual reconfiguration of the Flood Frog during the installation, an LCD display and four buttons are provided onboard; they can be used as input and output interfaces to the device.

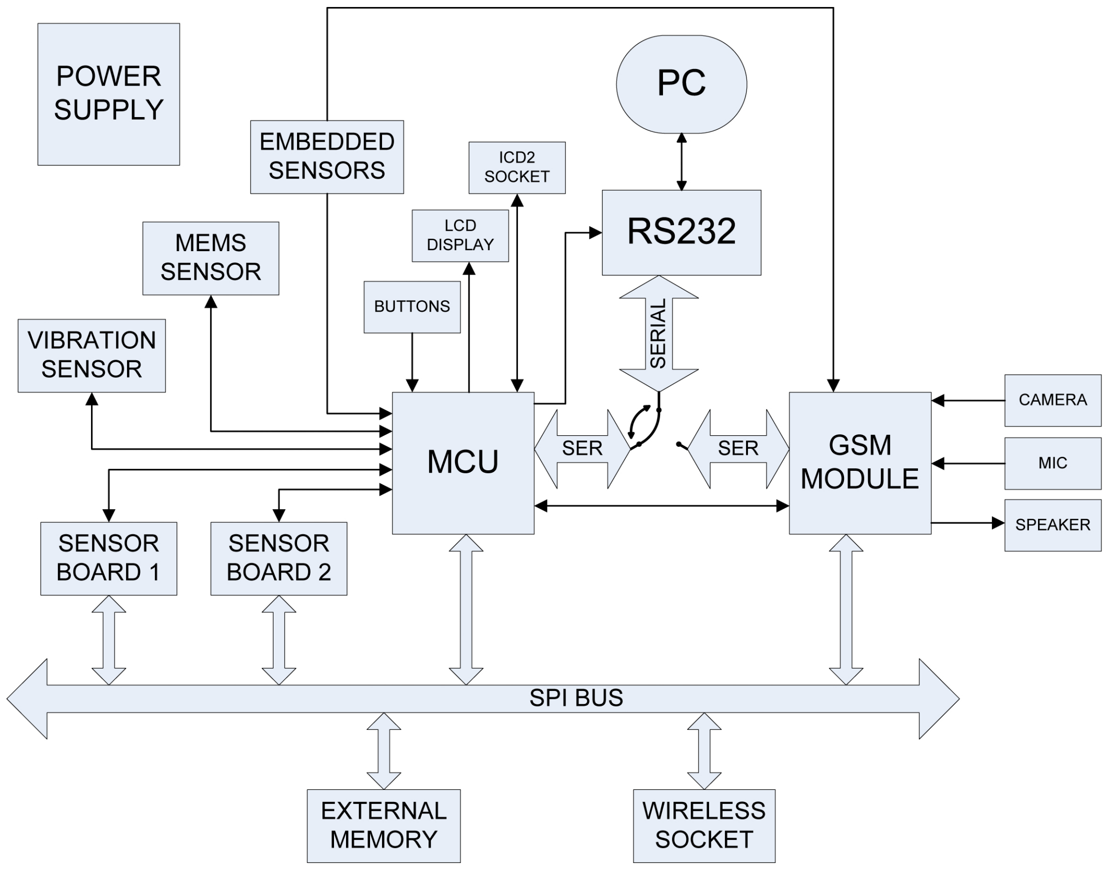

Figure 3. Overview of hardware architecture.

The Flood Frog is an embedded device built from the ground up, and its unique requirements necessitate the development of a custom operating system. The four main objectives of the software development are:

- small size
- reliability
- efficiency
- power awareness

The main goal of the software development is to create the smallest and least complex Operating System (OS) that can execute all required operations. Small size (in terms of lines of code) is desirable due to the limited memory space of the MCU; the need for low complexity arises from the dependability requirements. Simple code is typically more reliable and fault tolerant. Troubleshooting and maintenance of low complexity code is easier, and any software faults are likely to be detected and repaired. Last but not least, the software has to be power aware.

The design has to take into account the limited energy available to the device, and then implement efficient software techniques to reduce computation and keep the device in sleep mode for as long as possible. This can be achieved not only by writing efficient code but also by manipulating the hardware capabilities, e.g., placing the peripherals in sleep or off state when they are not required. The MCU has an on-chip flash memory of 144 KB, which is large enough to constrain the size of the code to an infeasible extent. To keep the possibility of updating the system, a bootloader has been implemented for remote firmware update. This additional component occupies a small fraction of the available memory. As in any embedded device, the interaction between hardware and software is very high in the Flood Frog, which facilitates the use of energy-efficient features of the hardware, such as sleep or low power modes.

The Flood Frog is a real time device, but due to the powerful onboard computational unit, there is no need to implement multitasking to satisfy the time constraints. An added benefit of sequential code over parallel implementation is dependability, which results from the relative ease of troubleshooting a single flow of execution. The only task that could require a fast real time reaction is the vibration alarm; in that case, the Flood Frog has to be switched on as fast as possible in order to avoid loss of information.

The coding was carried out with the Microchip MPLAB© integrated development environment [2]. This suite supports programming in two languages, Assembly and C. The majority of the Flood Frog software was written in C, which as a high-level language, provides for easier development. The included C compiler is the GNU GCC 3.3 [5] with a number of additions that support custom features of the MCU.

The software on an embedded microcontroller is typically simple. The basic sequence of operations is to read the input, elaborate the data, then write the output. Depending on the computational capabilities and memory space of the chip, programs with different degrees of complexity can be written and executed. The Flood Frog MCU is a 16-bit 30 MIPS processor that is more powerful than an Intel 80486 (20 MIPS [6]) chip. This computational capability was one of the motivating factors for development of sequential code. Parallel implementation may have reduced the response time of the system, but the less complex sequential code meets the timing requirements, which vary from one application to another. For example, the response time for flood monitoring is on the order of seconds. In contrast, vibration detection for structures poses a hard timing requirement on the order of milliseconds.

The OS is comprised of several components, each of which controls a portion of the device, such as the GSM module, the clock, and the serial port communication. In a fashion similar to Application Program Interface (API) development, each component can be considered as a constituent library of the OS; all of them are used in the execution core, creating the overall OS. This partitioning of functionality is intended to facilitate modular development of the software.

The Flood Frog is a monitoring device that can be used in either time-driven or event-driven fashion. As a time-driven device, the data collected by the sensors is recorded periodically, and the device remains in sleep mode unless it is recording data or an exception occurs. The recorded data is then compared with a prespecified threshold, and alarms are triggered as necessary. The reporting of data to the external world is also carried out periodically.

In event-driven mode, the device wakes up in response to prespecified events, the occurrence of which is detected by the sensors. The sensors, which can be embedded (e.g., flood, tampering, vibration) or external, are connected to input pins of the MCU. An interrupt is configured for each pin, and can cause a wake up of the device and the activation of its interrupt service routine (ISR).

The following interrupt sources have been implemented in the current prototype of the Flood Frog.

- timer#1
- timer#4
- flood
- missing floater
- tampering
- vibration

Timer#1 (denoted as such by the MCU) is a 16-bit timer that is used as a counter for the real time clock. It is initialized with a period of 1 second and utilizes the 32KHz oscillator (i.e., ultra low power crystal). At each timeout, it triggers its interrupt service routine, which first updates the real time clock (i.e., current time and date), then updates the counters for heartbeat and data log. If either variable goes to zero, the ISR pushes the related task (heartbeat or data log) onto the scheduler stack. The final job of this handler is to check if any button has been pressed, indicating that the manual operation task should be scheduled.

Timer#4 is a 32-bit timer initialized with a 1 msec period. It is used to implement a *Wait(msec)* function that introduces delays into the software execution flow, allowing the hardware peripherals to reach steady state before resuming execution.

The three MCU pins connected to the flood, floater presence and tampering reeds are constantly monitored by the internal hardware of the MCU; if they change state, an interrupt handler flags the source and pushes the corresponding alarm task onto the scheduler stack. These are sporadic events that happen suddenly, and this is the most efficient way of monitoring them.

The last interrupt source is the vibration sensor. Once the oscillation is detected (i.e., it exceeds a predefined threshold) the analog signal is immediately sent into a delay line that provides enough lag to allow the sampling circuit to be switched on. Meanwhile, the MCU senses the interrupt and invokes the appropriate ISR, which pushes a vibration task onto the scheduler stack. This task will manage the sampling and storing phase; moreover, it will trigger an alarm when required (e.g., an earthquake is in progress).

As described above, the ISRs for the various interrupt sources are quite similar. In every case, an ISR is invoked as quickly as possible in response to an event. For the Flood Frog, vibration is the most critical phenomenon being monitored; hence, its interrupt is given the highest priority.

The C language does not support multi-thread architecture, so only one task can be executed by the OS at any given time. As a result, the code has a sequential structure and does not incorporate parallelism. This choice eliminates any problems that could arise due to the sharing of resources; at any time there is only one process running and it can use the resources (e.g., GSM module, peripherals, serial channel) it needs.

The entry point to the execution core is the *main()* block. The Flood Frog is an autonomous device, and must retain minimal functionality even when in sleep mode. As a result, the software must run continuously, but keep the system in low power mode as often as possible. To support continuous operation, the *main()* block includes an infinite loop, in which all operations are performed. In the event of an exception, a complete hardware and software reset (i.e., reboot) of the device may be necessary for returning it to a safe and predictable state.

Figure 4 depicts the software state diagram of the device. The software execution flow has a single entry point, where the software begins initial execution and to which the software returns in the event of system reset. Immediately afterward, the source of the system reset is determined. Depending on the source, an appropriate task is pushed onto the scheduler stack. The purpose of this task is to record and inform the external world of the cause of the exception.

The events that result in a system reset can be divided into three main categories:

- manually induced,
- hardware exceptions,
- software exceptions.

The first case is the generation of a master clear signal by the ICD2 interface. This line is raised to start execution of the system software. The hardware exceptions include low battery voltage events. The software exceptions comprise math errors, stack errors and address errors.

At the entry point to the software, after the reset source is determined, an initialization routine is performed to prepare the various hardware components for operation. The initialization routine contains the following tasks.

- initialization of the bootloader
- initialization of the I/O ports
- configuration of the interrupts
- configuration of the clock and timers
- initialization of the sensors
- retrieval of data from EEPROM
- pushing the heartbeat task onto the scheduler stack
- preparing the hardware for sleep mode

The I/O ports are configured as tristate (high impedance state), so they do not drain current when not in use. Depending on the application, all or some of the interrupts are enabled, and the same is done for the timers. The sensors are initialized, and the EEPROM is queried to retrieve the information necessary for communication. Before preparing the hardware for sleep mode, a heartbeat task is pushed onto the scheduler stack; this task is self-triggered, meaning that when its timer expires and the ISR is executed, it automatically schedules the next execution.

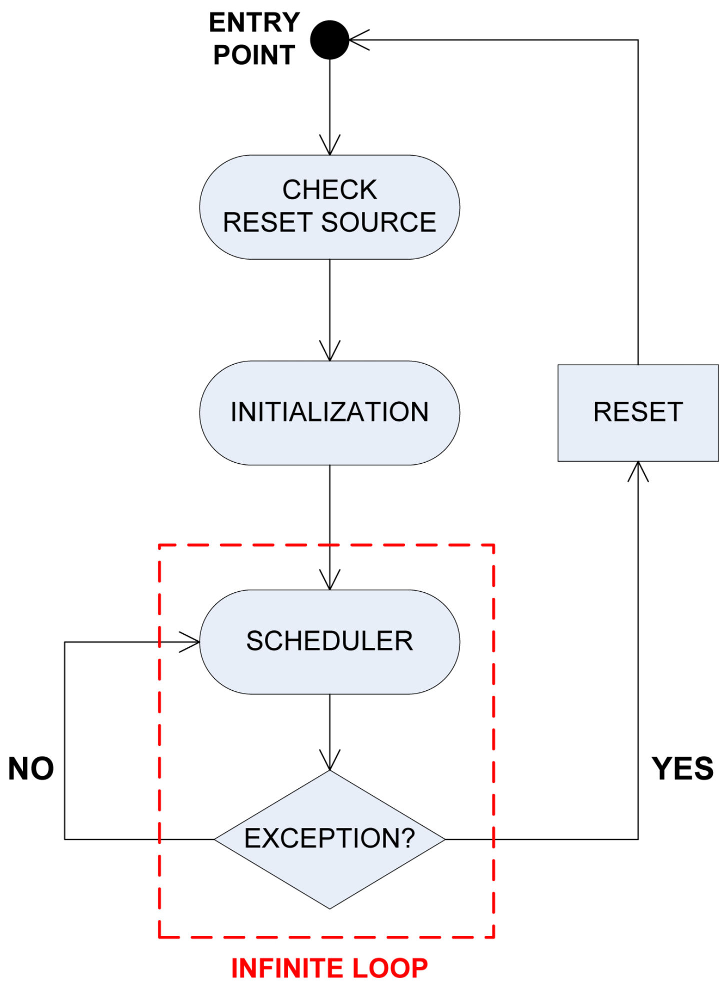

Figure 4. Software state diagram.

The infinite loop, which is implemented as a *while(1)* instruction, contains all operations performed by the device. A complete execution cycle of the software is comprised of the following phases.

- sleep period
- manual button check
- scheduler check

The initialization block of the execution core prepares the hardware for sleep mode. Subsequently, the first operation of the infinite loop is to place the Flood Frog in sleep mode. When its sleep timer expires, the device wakes up and checks the onboard buttons. If any of them has been pressed, a manual task is pushed onto the scheduler stack. The stack is implemented as an array where the ready processes are placed, ordered by priority. The highest priority task is always at the head of the queue, which guarantees that it will be executed first. If the scheduler stack is not empty, the first task is popped and executed; when it is completed, the next task, if any, is popped. In case the queue is empty, the device is returned to sleep mode. This design results in very simple and computationally efficient scheduler.

As depicted in Figure 4, the only exit point from the infinite loop is the occurrence of an exception. Figure 4 also illustrates the reset block, which resets the device when something goes wrong. In the event of a failure, the program is restarted from the entry point to ensure clean execution. The cause of the system reset remains flagged to allow it to be checked immediately after the reboot. This operation is represented by the check reset source block in Figure 4. The following pseudocode illustrates the execution flow of the system.

```
main() //entry point
{
   ResetSourceCheck();
   Initialization();
   SleepPreparation();
   while(1) //infinite loop
   {
      Sleep();
      ManualButtonCheck();
      while( ReadyTaskQueue != empty )
      {
         PopAndExecute();
      }
      SleepPreparation();
   }
}
```

The MCU allows for self-update of the internal firmware; while the system is running, the contents of flash memory can be modified by user code. The flash memory is organized as a table. The basic update procedure involves setting up a table pointer, then continually writing to memory until all of the new code is stored. The new firmware flows from the MCU's serial port, instead of the ICD2 interface used to flash the MCU the first time.

The self-update option has been implemented in the MCU's hardware to allow onboard programming without the use of the ICD2 hardware. In this way, the generic device can be updated by the final user avoiding factory recalls. The only requirement is the availability of an onboard serial port to supply the new data to be written. The original source of the data is irrelevant, as long as it arrives through the serial port in the correct format. As the system is designed to be wireless, a wired connection to the serial port is not available and radio-based remote update is necessary. The hardware allows direct communication between the GSM module, which receives the remote updates as direct data call, and the MCU.

Serial communication with the GSM module requires a handshake procedure to be implemented by the software. After this handshake takes place, the GSM module waits for the incoming data call that will carry the firmware update. Unlike an FTP transaction, a data call does not rely on the GPRS infrastructure, which may not be available in remote locations where the Flood Frog will be deployed. An incoming call is chosen over an outgoing call, as it eliminates the necessity of storing static phone numbers in the device, and assigns control of the update to the remote entity rather than the device.

The self-updating feature can achieve significant reductions in the cost of maintenance, but it can also introduce vulnerability in the form of an update failure, or a fault in the new code. Failure of a remote update can be due to any of a number of reasons, including a dropped call due to weak signal coverage. An unsuccessful update can lead to unpredictable behaviour of the system, as it is left in an undefined state. The real danger is a crash that eliminates the possibility of further remote updates. In this case, the device will be unable to operate any longer, necessitating a costly site visit.

To increase dependability, a bootloader has been implemented in the software. At the reset instant, the code checks a flag that determines the subsequent execution flow. The two possibilities are execution of the main routine, or the bootloader code. The former implements all the features of the Flood Frog, and is executed during normal operation. The bootloader includes the code to set up serial communication, answer an incoming data call and write to the flash memory. Execution of the bootloader serves as an emergency mechanism that enables the establishment of a connection and update of the firmware.

The bootloader code is executed at each machine reset. In the rare event that an update becomes necessary, the user code must force a system reboot. If the bootloader timeout expires before an incoming call is received, the main code is executed as usual. In case a call is received, and the arriving file is recognized as correct, the update procedure begins. The data arrives serially, and is immediately written to memory. The MCU lacks sufficient memory to store the entire firmware, so it has to be written directly to flash memory. As the writing operation is relatively slow, flow control is implemented in the serial communication routine. If the update terminates correctly, after the device is rebooted, it will run the new firmware from that point onward. If the update is aborted, a flag is set in the EEPROM to signal this failure. At reset, the bootloader will check this flag, detect the failure, and wait for another update, rather than executing the main routine.

It is clear that the main code and the bootloader code should reside in separate and independent memory areas; moreover, the bootloader block must be protected from accidental overwrites, as it is vital for correct and continual operation of the device. The physical separation of routines necessitates a larger memory. This does not burden the design, as the bootloader code is much smaller than the main routine, and the MCU has a reasonable amount of memory.

4 Findings

The Flood Frog is composed of more than 250 electronic components, the majority of which are resistors, capacitors and connectors. The expense incurred for building a prototype is mainly due to the cost of components, PCBs, and manufacturing.

The cost of components includes the cost of all parts mounted on the PCB, for a total of approximately \$190. The MCU and GSM modules, at costs of approximately \$10 and \$100 respectively, are responsible for the majority of the cost of components. The high cost of the GSM is not surprising, as it embeds a complete quad-band mobile phone with Internet connectivity, an internal microcontroller and a Python interpreter. GPS can be purchased as an added feature, enabling a wide range of additional applications.

Due to the need for housing a large number of components in a very limited space, the Flood Frog's PCB uses traces with a minimum width of 0.2 mm, which necessitates industrial printing of the board. For individual units of the device, this is an expensive procedure, but the cost per unit decreases dramatically for larger batches. Similarly, the cost of manufacturing the device comprises a large fraction of the overall cost, especially for individually handcrafted units. The manufacturing of larger batches can be automated, leading to significant cost reduction.

Table 1 presents cost estimations for various production volumes. The final cost, even for single unit production, is significantly lower than existing monitoring solutions. Even for small structures such as low water bridges located in rural sites, this cost is negligible as compared to the overall building budget. The reduction in maintenance costs achieved over comparable monitoring solutions further underscores the savings.

Table 1. Cost of production for various volumes.

	1 UNIT	100 UNITS	1000 UNITS
Components	190	178	155
PCB	63	12	7
Manufacturing (% of total)	187 (42%)	80 (29%)	40 (19%)
TOTAL	440	270	202

The raw power consumption of the MCU depends on the frequency at which it operates; the higher the speed, the greater the energy required. The MCU needs an oscillator that works as the system clock. The nominal frequency of this component affects the computational speed, and consequently the power consumption. For that reason, the MCU has four different oscillator sources available:

- primary external oscillator (8 MHz)
- secondary external oscillator (32 KHz)
- fast internal RC (8 MHz)
- low power internal RC (512 KHz)

The primary oscillator can use an internal Phase Locked Loop (PLL) as a multiplier to increase the clock frequency and achieve better performance. The secondary oscillator is designed specifically for low power operation with a 32 KHz crystal. Both of these oscillators are external, and much more accurate than the internal ones. The fast internal RC oscillator cannot be multiplied by the PLL. The last source is a low power internal RC oscillator that uses the slower crystal for power saving purposes.

Table 2 shows the power consumption of the Flood Frog while utilizing different oscillators. The first four rows correspond to the available crystals in the device, as described earlier. GSM_PWR and AUX_PWR are the power sources that supply, respectively, the GSM module and the peripherals. The RUNNING column is the current drained when the MCU is computing, while the SLEEP column indicates power saving mode; the first is measured in mA and the second in μA.

The XTAL4xPLL @8MHz mode is the most computationally powerful. The 8MHz clock is multiplied by 4 to obtain a 32 MHz input frequency, which results in the highest power consumption in both running and sleep modes. In order of descending frequency, the next oscillator is the fastRC @8MHz, which reduces the current by a factor of 4, due to the change in clock speed. The lowRC @512KHz further reduces the clock speed, and consequently the power consumption, resulting in a current drain of only a few mA (or μA in sleep mode). Lastly, as expected, the XTAL @32KHz oscillator results in the lowest power consumption while the MCU is in running mode.

The two center rows in Table 2 show the case when either the GSM module or the AUX peripherals are on. The power consumption for this case is measured only when the MCU is running, as it is pointless to switch on peripherals when the controller itself is not running. From this data, the current drained by the GSM module (in power saving mode) is calculated to be 36.2 - 33.4 = 2.8 mA. Similarly, the AUX peripherals drain 56.2 - 33.4 = 22.8 mA.

The last row of Table 2, presents the case when the MCU is running but in idle state, i.e., not computing. This condition occurs when the Flood Frog has to stay awake for a period of time waiting for remote update. In this case, the Flood Frog has to be ready to react, but the slowest possible oscillator suffices.

Table 2. Power consumption for different oscillators.

CLOCK SOURCE	GSM_PWR	AUXA_PWR	RUNNING (mA)	SLEEP (μA)
XTAL4xPLL @8MHz	off	off	33.4	30
XTAL @32KHz	off	off	1.9	25
fastRC @8MHz	off	off	9.6	20
lowRC @512KHz	off	off	3.6	9
XTAL4xPLL @8MHz	ON	off	36.2	-
XTAL4xPLL @8MHz	off	ON	56.2	-
XTAL @32KHz	off	off	1.0 (idle)	-

The powerful 32 MHz mode is used for heavy computation, for instance in case of digital signal processing or the transfer of large volumes of data through the serial port. The serial port can be configured at many speeds, but to run at the highest speed of 115 Kbps, the MCU also needs to operate at the 32 MHz mode to be able to support the communication.

Determining the real current drained from the batteries requires examination of the frequency of various tasks performed by the controller. The degree of activity of the Flood Frog is directly related to its energy consumption.

The Flood Frog operations are divided into three tasks:

- daily operation
- daily heartbeat
- weekly data report

The first task takes into account energy required on a daily basis to maintain the Flood Frog in a low power consumption state. Even in sleep mode, which is the mode with the lowest power consumption as explained earlier, the device has to control all alarms and be ready to react. That implies being powered 24 hours a day. This level of daily activity for the Flood Frog implies an average energy consumption of approximately 0.3 mAh/day.

The second task includes the daily activities associated with maintaining the heartbeat signal. The startup includes switching on the system and send an SMS to communicate that the device is functional. Every day, after sending the SMS, the device is in an idle state for 10 minutes, waiting for incoming communication of a firmware update. The firmware update happens very sporadically, and as such has been excluded from calculations of power consumption. After this idle period, the device disconnects from the GSM network and goes to sleep again. Overall, the aforementioned heartbeat activities consume approximately 5.85 mAh/day, which is considerably higher than the daily power consumption associated with sleep mode alone (0.3 mAh/day, as explained in the previous paragraph). The difference is mainly due to power consumed during the waiting window (4.1 mAh/day), and to a lesser extent, the GSM enrollment (1 mAh/day).

The third and last task is the weekly schedule, which is similar to the daily section, but instead of the waiting window there is a data communication phase, when the Flood Frog is sending all of the information collected during the week. SMS, e-mail and FTP files are sent out twice in a row to increase the reliability of communication. The total energy for this phase is 1.15 mAh/day, which is a relatively low value because it is actually a weekly consumption, normalized over a daily basis.

The unattended field life can be estimated using the aforementioned power consumptions, which are measured from the prototype, and Equation 1. In this equation the variable "RealBatteryEnergy" represents the actual energy stored in each of the cells used, as opposed to the nominal value. For the D-cells used in the prototype, 11 Ah is a realistic estimate of this value.

$$\begin{aligned}\text{BatteryLife} &= \frac{\text{RealBatteryEnergy}}{\text{DailyOperation} + \text{DailyHeartbeat} + \text{WeeklyDataReport}} \\ &= \frac{11000}{0.299 + 5.847 + 1.153} \simeq 1507 \ \text{days} \simeq 4.13 \ \text{years}. \end{aligned} \tag{1}$$

The estimated battery life of 4.13 years verifies that the goal of low power consumption, and consequently, long unattended field life has been attained. This figure was calculated with the inclusion of a daily waiting window of 10 minutes. This waiting window, which is in anticipation of a rare event such as firmware update or unexpected maintenance, comprises approximately 70% of the daily power consumption. The waiting window has been designed to facilitate rapid reconfiguration of the device in the event of a failure or change in application requirements. For non-critical applications, the waiting window can be executed every other day rather than on a daily basis. This reduces the overall energy consumed by the waiting window by a factor of two, to 2.08 mAh/day, which consequently reduces the energy consumption from 5.84 mAh/day to 3.763 mAh/day. In this case, the unattended field life is extended to 5.77 years, as calculated by Equation 2. This simple trade off between energy consumption and flexibility of use, accomplished by modifying the schedule of operations, adds 1.5 years to the unattended field life of the device.

$$\begin{aligned}\text{BatteryLife} &= \frac{\text{RealBatteryEnergy}}{\text{DailyOperation} + \text{DailyHeartbeat} + \text{WeeklyDataReport}} \\ &= \frac{11000}{0.299 + 3.763 + 1.153} \simeq 2109 \ \text{days} \simeq 5.77 \ \text{years}. \end{aligned} \tag{2}$$

The window mechanism is the safest way to guarantee timely upgrades and maintenance for the Flood Frog, but once the device is considered reliable, as a result of comprehensive and successful field testing, the waiting window can be scheduled *on demand*. When communication with the device is required, an SMS can be sent to open the waiting window at a specified time, so the Flood Frog can be called and upgraded. Daily enrollment with the GSM network is still required, in order to receive the SMS and parse it to identify the eventual communication request. Disregarding the power consumed by the very sporadic event of the waiting window, the daily power consumption is reduced to 1.681 mAh/day.

$$\begin{aligned}\text{BatteryLife} &= \frac{\text{RealBatteryEnergy}}{\text{DailyOperation} + \text{DailyHeartbeat} + \text{WeeklyDataReport}} \\ &= \frac{11000}{0.299 + 1.681 + 1.153} \simeq 3511 \ \text{days} \simeq 9.61 \ \text{years}. \end{aligned} \tag{3}$$

Equation 3 demonstrates that the unattended field life of the Flood Frog, assuming an on demand waiting window, is 9.61 years. This is an impressive result, as the field life now exceeds even the physical life of a chemical cell. Due to concerns over the aging of chemical substances, disposable batteries are typically sold with an expiration date of no more than 6 years after the date of manufacture. In this scenario, the Flood Frog may be able to utilize the entire life of a standard battery pack, which indicates that it is prudent to design the optimal task schedule based on the available battery life. Information about the battery status is communicated daily through the heartbeat message, ensuring that unexpected changes in battery health (i.e., output voltage) will be detected in a timely fashion.

One fundamental feature of the Flood Frog is the wireless communication capability. Eliminating the need for wires significantly reduces the installation cost, and facilitates the use of a sealed enclosure for the device. The radio-based long range communication utilizes the existing GSM mobile infrastructure to exchange information with the external world. The quad-band GSM module allows worldwide compatibility. Acquiring a local SIM card allows sending an SMS or using GPRS connection to access the Internet for only few cents. The charges are usually based on the actual traffic exchanged, and the fixed fees are negligible. For the first field deployment of the device, on Bridge A6531 in Osage Beach, MO, a Cingular SIM card was chosen, as this provider has the best coverage in the selected location and offers pay per use phone plans. Table 3 summarizes the cost of mobile services with a Cingular prepaid plan purchased in 2006. Other service providers charge similar fees.

Table 3. Mobile network service costs.

ACTIVITY	COST (US cents)
Sending SMS	10/message
Sending e-mail	1/KB
Send/Receive FTP	1/KB
Voice/Data Call	25/min

Using the information in Table 3 in conjunction with the illustrated data traffic, the annual service charges can be calculated using Equation 4.

$$\begin{aligned}\mathrm{YearlyCost} &= (365 * \mathrm{SMSCost}) + 52 * 2 * (\mathrm{SMSCost} + \mathrm{emailCost} + \mathrm{FTPCost}) \\ &= (365 * 0.1) + 52 * 2 * (0.1 + 0.05 + 0.3) \simeq \$80 \qquad (4)\end{aligned}$$

The calculation in Equation 4 is based on the assumption that an e-mail uses 5 KB of traffic, while an FTP transaction uses 30 KB. The GSM module limits the body of an e-mail (without attachments) to 512 characters (i.e., 0.5 KB). Adding the GPRS and TCP overhead, the total traffic exchanged while sending an email increases by a few KB, so 5KB is a generous estimate for the email case. For an FTP transaction, a 20 KB file is assumed, plus 10 KB overhead due to GPRS and TCP, which is a generous estimate that considers the worst case scenario, where multiple retransmissions are required.

The estimated cost of $80 per year is negligible as compared to the cost of other monitoring techniques, or damages incurred by flooding. If the GPRS facility is not available, another way to exchange raw data is a data call. Data calls are considered voice traffic, which costs 25 cents per minute. Due to the small volume of data exchanged and the adequate GPRS coverage typically available, this cost is incurred so rarely that it can be considered insignificant.

The Flood Frog was installed on Bridge A6531 in Osage Beach, MO in November 2006. Figure 5 shows the box affixed to the bridge pier, where the yellow cable is the probe to measure the water level. The device detects flooding and measures water level, temperature, battery level, and inclination of the structure along three axes. The exposed and outdoor location was intentionally chosen to to test the robustness against natural elements. The only protection is the original case of the device. Since being installed, the Flood Frog has delivered the heartbeat message through SMS and e-mail daily, and has uploaded the acquired data to the FTP server. To increase the dependability of information delivery, it is currently using two recipients for each communication channel.

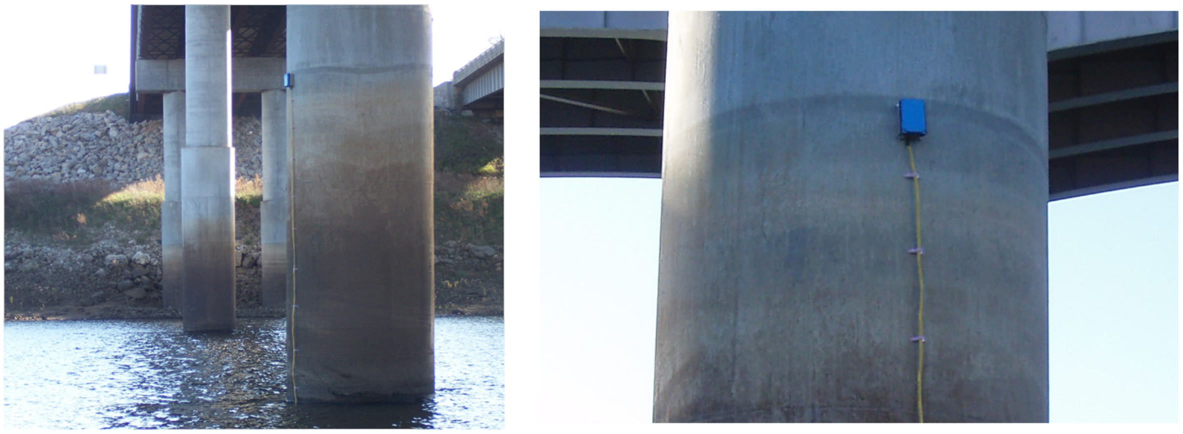

Figure 5. Installation of the Flood Frog on Bridge A6531 in Osage Beach, MO.

In contrast to other existing monitoring solutions, the Flood Frog is operating completely autonomously and wirelessly. The only source of power is the battery and all communication flows through the radio link. The overall cost of the device is less than \$500, which is negligible as compared to the cost of the monitored structure or to traditional (i.e., wired) monitoring systems. The monitoring is performed in real-time, allowing a prompt reaction to critical and dangerous events such as flood and structural stress.

5 Conclusions

Structural health monitoring is widely recognized as an important aid for improving the safety and reducing the maintenance cost of infrastructures. At present, only a small fraction of existing structures incorporate automated monitoring. Apart from research applications, the investment associated with installation of autonomous monitoring systems is considered a justifiable expense only in structures with particular strategic, historical or economic importance.

Analysis of the costs associated with autonomous structural health monitoring indicates that design, installation, software customization and calibration of the system comprise the majority of the expense. Moreover, the variety of sensors required, as well as the necessity for related electronic components limit the scalability of the existing systems.

The specific contribution of this research is the development of an autonomous low-power wireless device designed to achieve dramatic reductions in both equipment and installation costs. It incorporates a variety of embedded sensors, is battery-powered and communicates using the GSM/GPRS mobile phone network, eliminating the need for cables of any type. The standard battery pack included in the waterproof case allows for a typical unattended field life of 3-4 years, although a much longer life can be achieved by fine tuning the frequency of operations. Embedded in the device are temperature, water level, tilt and acceleration sensors. A digital camera, as well as supplemental analog and digital sensors, can be optionally added without requiring additional conditioning electronics or power sources.

The data collected by each sensor, any alarms triggered, and software anomalies detected during operation, are written to the internal memory and automatically delivered to a number of recipients through SMS messages, e-mail, and FTP file upload. Parameter adjustment and software upgrades can be carried out remotely, reducing the cost of maintenance.

The Flood Frog is ready to be networked through the addition of lightweight sensor nodes that expand the monitoring capabilities of the system. A local wireless sensor network would allow more accurate measurement of quantities, as the sampling points are distributed along the structure, which can be arbitrarily large, especially for bridges and buildings.

The cost reduction achieved by the Flood Frog has the potential to expand the practice of structural health monitoring to a significantly higher number of existing and new infrastructures. This improvement will increase safety and reduce the cost of operations, by facilitating real-time monitoring and yielding a more efficient maintenance schedule that extends the useful life of a wide range of infrastructures. Additionally, its general design allows easy adaptation to a wide range of alternative monitoring applications.

6 Recommendations

Since the Flood Frog has been designed as a general and multipurpose monitoring system, future developments and applications are virtually unlimited. The device is currently battery powered, but the batteries can be supplemented with solar energy by using solar panels and a transparent box cover. This addition is illustrated in Figure 6. As depicted in this figure, the energy from the solar panel is stored in a super capacitor that feeds the remaining circuitry. If sufficient solar energy is harvested, and the Flood Frog spends most of the day in sleep mode, solar energy can replace a significant portion of the energy currently provided by the batteries. This enables the use of smaller batteries, or in the case of a wireless sensor network for monitoring, serves as the sole source of power for peripheral nodes with limited computation and communication capabilities. Eliminating batteries will result in the avoidance of problems arising from the aging of chemical components, or fluctuations in their efficiency due to changes in temperature.

In the current Flood Frog prototype, long range communication is through the GSM mobile network; this choice has been made due to its worldwide availability. GSM provides acceptable coverage almost everywhere, but a limited number of mobile carriers use the CDMA protocol instead, which is a completely different and incompatible system. To enable the Flood Frog to communicate through this platform, a CDMA module can be provided onboard, which will equip the device with the two most popular mobile protocols in the world.

Large-scale monitoring of an infrastructure or area can be performed with a wireless sensor network in which the Flood Frog serves as the base station. Peripheral sensor nodes collect data from their immediate vicinity, and send it to the base station for processing. The nodes should be able to self-organize with a power efficient routing protocol. In this scenario, it would be possible to use the solar harvesting mentioned above to eliminate the batteries and lengthen the node life. The

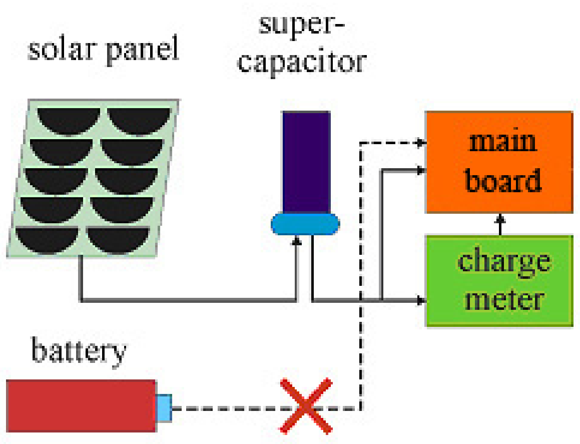

Figure 6. Use of a super capacitor for harvesting solar power.

nodes should also be inexpensive, to allow a large number of them to be deployed. The proposed architecture is illustrated in Figure 7.

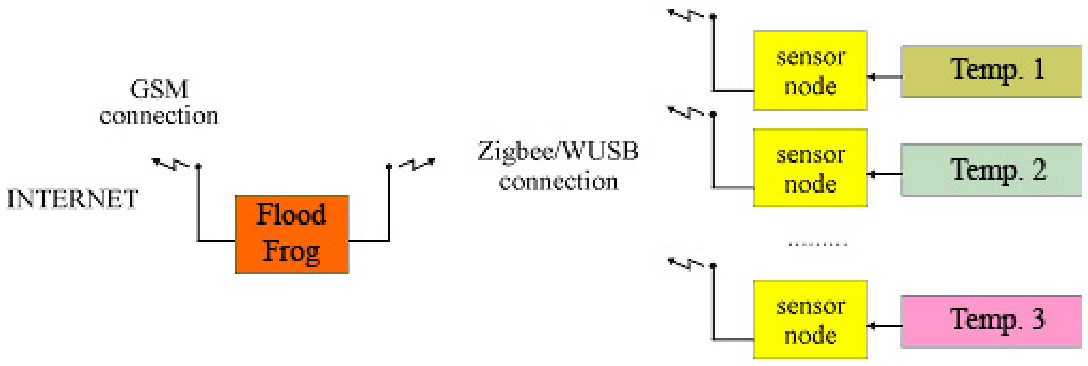

Figure 7. Wireless sensor network based on the Flood Frog.

In the network of Figure 7, several nodes are used to measure the temperature at various points of the monitored site; all of them transmit data through a radio link (e.g., ZigBee or WUSB) to the base station (i.e., the Flood Frog) that serves as an aggregator. The base station periodically sends the aggregate data to the external world by using its long range communication capabilities (SMS, e-mail, FTP). The addition of a wireless sensor networks would greatly enhance the scalability of the system.

References

[1] Electrical resistance tomography for subsurface imaging. U.S. Department of Energy Office of Environmental Management Office of Science and Technology, June 2000.

[2] Microchip home page. `http://www.microchip.com/`, 2006.

[3] BOPLA GmbH. Protection classification IP 68. `http://www.bopla.de/english/technische_informationen/ip68.html`.

[4] European Space Agency. Civil protection assistance. `http://www.esa.int/esaEO/SEMQFF3VQUD_environment_0.html`, July 2004.

[5] Free Software Foundation. GNU C Compiler Home Page. http://gcc.gnu.org/, 2006.

[6] Intel Corp. MIPS processors comparison. Microprocessor Quick Reference Guide, 2006.

[7] Jerome P. Lynch et al. Advanced wireless structural monitoring: Past, present and future. The John A. Blume Earthquake Engineering Center, Issue 28, 2001.

[8] M. Maroti et al. Shooter localization in urban terrain. *IEEE Computer*, 37(8):60–61, Aug. 2004.

[9] National Oceanic and Atmospheric Administration. Flood damage in the United States, 1926-2003: A Reanalysis of national weather service estimates. `http://www.flooddamagedata.org/`, 2005.

[10] NOAA - National Weather Service. Flash flood monitoring and prediction (FFMP). `http://www.nws.noaa.gov/mdl/ffmp/index.htm`, Mar. 2006.

[11] R. Szewczyk et al. Habitat monitoring with sensor networks. In *Communications of the ACM*, volume 47, pages 34–40, June 2004.