A high performance, low power computational platform for complex sensing operations in smart cities

Jiming Jiang, Christian Claudel

PII: S2468-0672(16)30017-7
DOI: http://dx.doi.org/10.1016/j.ohx.2017.01.001
Reference: OHX 3

To appear in: HardwareX

Please cite this article as: J. Jiang, C. Claudel, A high performance, low power computational platform for complex sensing operations in smart cities, HardwareX (2017), doi: http://dx.doi.org/10.1016/j.ohx.2017.01.001

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.
A high performance, low power computational platform for complex sensing operations in smart cities

Jiming Jiang\textsuperscript{a}, Christian Claudel\textsuperscript{b},\textsuperscript{*}

\textsuperscript{a}King Abdullah University of Science and Technology, Thuwal, 23955-6900 Kingdom of Saudi Arabia
\textsuperscript{b}University of Texas at Austin, 301 E Dean Keeton St C1761 Austin, TX 78712 USA

Abstract

This paper presents a new wireless platform designed for an integrated traffic/flash flood monitoring system. The sensor platform is built around a 32-bit ARM Cortex M4 microcontroller and a 2.4GHz 802.15.4 ISM compliant radio module. It can be interfaced with fixed traffic sensors, or receive data from vehicle transponders. This platform is specifically designed for solar-powered, low bandwidth, high computational performance wireless sensor network applications. A self-recovering unit is designed to increase reliability and allow periodic hard resets, an essential requirement for sensor networks. A radio monitoring circuitry is proposed to monitor incoming and outgoing transmissions, simplifying software debugging. We illustrate the performance of this wireless sensor platform on complex problems arising in smart cities, such as traffic flow monitoring, machine-learning-based flash flood monitoring or Kalman-filter based vehicle trajectory estimation. All design files have been uploaded and shared in an open science framework, and can be accessed from \cite{1}. The hardware design is under CERN Open Hardware License v1.2.

Keywords: Wireless sensor network, Embedded system, Artificial Neural Networks

\textsuperscript{*}Corresponding author

Email address: christian.claudel@utexas.edu (Christian Claudel)
1. Introduction

1.1. Traffic congestion and flash flood monitoring

Traffic congestion is an increasing burden in many areas of the world. In the U.S.A. alone, traffic congestion caused a 101 billion dollars economic loss, 4.8 billion hours of cumulated delay and 1.9 billion gallons of wasted fuel in 2010 \[2\].

In most OECD countries, traffic congestion is estimated to cause 1% loss in its gross domestic product (GDP), that is, over 100 billion Euros annually for the European community. In most countries, the direct and indirect costs of traffic congestion are considerable, and are expected to become worse as global traffic demand is expected to greatly increase in the next decades.

One of the most promising solutions to address the traffic congestion problem is traffic flow control \[3\]. However, one of the prerequisites of traffic flow control is accurate traffic data, and accurate estimates and forecasts of origin-destination matrices, which drive the evolution of traffic demand over a transportation network. This data is usually not accurate enough for traffic control applications. Another critical issue in the operation of future cities is their resilience to natural disasters. Flash floods are one of the most common natural disasters, and typically cause more casualties than all environmental disasters combined. Unfortunately, mapping flash floods in real time using wireless sensor networks is relatively costly, and may not be considered by cities since floods are perceived as a non-recurrent issue, unlike traffic congestion. To make the solution more cost effective, we developed a dual traffic/flash flood sensor system that can sense both phenomena with no added marginal cost for the flood application. This system however requires some advanced processing techniques such as supervised machine learning and model-based sensor fusion, and thus requires the development of a new sensing platform capable of handling these computations. However, none of the currently available platforms allow us to have at the same time high computational performance, energy monitoring, and ability to operate unsupervised for extended periods of time, which requires the development of a new sensing platform capable of meeting these specifications.
1.2. Eulerian and Lagrangian Sensing

Traffic monitoring systems can be classified as fixed (Eulerian) or mobile/probe (Lagrangian) based systems [4]. The Eulerian system can be further classified as intrusive or non-intrusive based on the installation setup.

Intrusive systems require the sensor to be deployed directly underneath the road, for instance in a saw-cut hole or in tunnels under the surface [5]. Loop detectors, piezoelectric sensors and pneumatic tubes are the widely used sensors [6]. These technologies have been developed over several decades, and are widely adopted in commercial systems, including traffic monitoring systems in Singapore, Korea or in the USA [7]. These systems are however very expensive both to deploy and to maintain, since any installation or maintenance operation requires the traffic to be blocked.

In contrast, non intrusive systems do not require an installation underneath the road, which reduces the cost and speeds up deployment operations. Widely used sensors include microwave radar, video image processing (camera), and infrared device, ultra-sonic or acoustic sensors. However, most conventional non intrusive systems are still based on wired sensor network technology, that significantly increases costs [6].

Recent traffic wireless sensor network deployments have been tested in [8], [9], [10], though these sensor systems are only partially wireless (cables are needed to relay data from sensing stations to computer servers).

The last decade has witnessed a radically new approach to traffic sensing called Lagrangian (mobile) sensing, which relies on the data generated by vehicles themselves. Lagrangian sensing has become increasingly prevalent in modern traffic information systems, with some systems relying on both types of data to generate traffic estimates [11]. Probe vehicle data consists in velocity, position and time data generated by some vehicles typically using the cellphone network (GPRS, 3G/4G) [12]. While probe data is relatively accurate and has an extremely low marginal cost, many issues remain associated with this technology, in particular the low penetration rate of participating users, who have to share their location data. Issues such as user privacy, or the high power
consumption of the GPS (in cellphones) prevent the large scale deployment of such systems.

As in most urban sensing systems, deployment and maintenance costs are typically larger than the costs of the hardware. To minimize deployment costs, we require the system to be fully wireless, that is to form a wireless mesh network, and to harvest its own power.

Given all the above constraints, the proposed traffic sensing system requires an energy efficient computational platform capable of handing a high computational load locally, while providing long distance communications and allowing operations over very long time frames (in years). The platform also needs to harvest solar energy as efficiently as possible. This combination is, to the best of our knowledge, not commercially available to date.

1.3. High performance wireless sensor networks

Wireless sensor networks (WSNs) have emerged as a possible solution for urban monitoring applications. They typically consist of sensing devices that can automatically construct a wireless network based on a predefined protocol. Because of their computation, communication and sensing capabilities, WSNs have been developed for a large number of applications, for instance, in environmental monitoring [14], health surveillance [15], human presence detection [16], and position discovery [17].

One of the main issues arising in the operations of WSNs is power management. To reduce power consumption, the computational capabilities of the nodes are typically very low, and the nodes merely act as sensors and wireless relays, with little or no processing of the measurement data done onboard. For example, in the context of traffic, the system [10] is based on a MICA2 node, and the processing unit is an 8bit ATmega128 microcontroller running at 7.4Mhz; [18] uses a commercial TeloSB node running on 16Mhz MSP430 controller. These systems do not have the capability to run complex algorithms, which are handled at the sink or computer server levels.

In contrast, our system requires advanced algorithms to run onboard the
nodes. As an example, the processing of traffic data involves mixed integer linear optimization problems with tens of variables and hundreds of constraints. The flash flood data is processed in parallel using Artificial Neural Networks that require tens of kB of memory, and advanced computational capabilities. All these constraints require a novel, high performance solar powered node to be designed.

Though some commercial WSN platforms ([19], [20], [21]) can provide high performance computation, none of them satisfy our application as a competent platform. Based on our requirements, such a platform should also have:

1. long range reception, to support data communication between sparsely deployed nodes in an adverse (urban) environment
2. high efficiency in energy harvesting to maximize the lifetime and the coverage of the system
3. high reliability hardware, with recovery capabilities
4. wireless firmware update to enable research operations, with frequent code updates over a large network, without having to recover and redeploy the nodes, which is a time consuming process

The lack of commercial platforms satisfying these constraints led us to design and develop a new hardware platform for flash flood/traffic sensing applications.

2. Computational requirements

Each node will have to carry out a number of computational tasks for both traffic flow and flash flood sensing:

- Machine learning-based vehicle classification [22].
- Traffic estimation and sensor fault detection.
- Machine-learning based temperature compensation for water level sensing, which is the focus of the present article.
3. Platform Architecture and Design

Our objective is to develop a low cost, low power platform capable of solving the aforementioned computational tasks in real time, for the sensor network architecture presented above. While higher computational performance increases both cost and power consumption, we choose to increase computational capabilities for all nodes for the following reasons:

1. Modularity, as we might add new applications to this sensor network (for example detecting the presence of water on the ground using Machine Learning). This approach reduces costs, simplifies the maintenance, and makes the complete system more fault tolerant than a system in which nodes play different roles with different sensor platforms, as in [23].

2. Bandwidth constraints, our fixed traffic sensors generate more than 70 points of data per second. Processing this data at the gateway level would require a bandwidth that is beyond the capabilities of commercial long range transceivers, particularly in a multi-hop communication setting.

3. Cost and power, as a faster 32 bit RISC microcontroller such as the Cortex M4 has comparable cost and power requirements (when operating with its lowest frequency setting) with more classical 8 bit microcontrollers such as the Atmega 1281.

In order to support a low cost, distributed, real-time and reliable traffic sensing system, the sensor platforms should have the following features:

- Low node cost, low deployment costs (installation on urban structures, no power and data cables required)
- Capable to some extent of self-recovering in case of software failure, and capable of periodic hard resets ([24])
- Capable of significant actuation for power management (solar energy harvesting, battery charging, reconfigurable clock frequency)
Advanced computational capabilities, with significant free memory to allocate relatively large matrices required by machine learning applications, and the capability for simultaneously running node and network energy optimization algorithms (energy-aware routing as in [25]).

- Capable of Over-the-air programming (OTA) for software update, since the code in a research setting is expected to be frequently modified, and the nodes are not easily accessible.

Following these requirements, we designed a hardware platform [26] [1], on which we are porting an operating system and a middleware to simplify programming. In comparison with other reported hardware platforms, the proposed platform has some specific features:

1. The frequency of microprocessor can be dynamically adjusted based on energy and computational time requirements, thus the platform can operate at the optimum frequency for different tasks.

2. A radio monitoring circuitry integrated to the platform allows a user to monitor both incoming and outgoing radio traffic to increase reliability or for debugging purposes.

3. A recovery circuitry to enhance the reliability.

These features are required to implement the proposed traffic/flash flood sensing vision. This architecture is simpler to deploy (owing to the distributed/onboard processing) than existing wireless sensor networks that are feeding their measurement data into input databases, and which require processing/estimation servers.

Figure 1 shows the block diagram of the hardware platform. Figure 2 illustrates the third generation hardware platform over the hardware development period (September 2012-May 2013). We now give a detailed description of this hardware platform by focusing on the following areas: processing unit, communication, data storage, power management and peripherals.
Figure 1: High level block diagram of the computational platform investigated in this article.

Figure 2: Third generation (March 2013). This platform has a 8-path integrated switch chip and separate USB ports (one for firmware upload, the other for radio monitoring).
3.1. Core element

The core component of a sensing platform is the microcontroller (MCU), which handles sensing (ADC and digital buses), computation and control. We selected for this application the STM32F407, a 32-bit ARM Cortex-M4 based microcontroller from ST since it satisfies the requirements described above and best balances the tradeoff among computation, RAM, power consumption and cost. We considered a wide range of microcontrollers, in which the ATmega1281 is on the low end (low performance, low power consumption) and the TI TMS570 on the high end (high performance, high power consumption). The microcontrollers in the low end are not able to provide with sufficient internal data RAM (8 KB), program memory (128 KB) and computational power (16 MHz). On the other hand, while the high end exhibits a fast frequency (180 MHz), they also have higher power consumptions and higher prices, which makes it unsuitable for an extensive, long term deployment. In contrast, the STM32F407 provides a comparable performance with TMS570 with only one third of the price; it is even less expensive than than a 8 bit ATmega1281. In addition, at its lowest frequency setting its power consumption is comparable to the power consumption of the ATmega1281.

The STM32F407 includes a 1 MB Flash memory and 196 KB of data RAM. It supports up to seventeen timers, a 24 channels ADC and two 12-bit DACs for peripherals.

On this proposed platform, the microcontroller is configured to have three Universal Asynchronous Receivers/Transmitters (UARTs), two Serial Peripheral interfaces (SPI), one I2C interface bus, one SMBUS interface for sensors and one Controller Area Network (CAN) interface. A 4 to 1 multiplexer is used for the UART extension, that allows a total 6 UART ports. Furthermore, a SDIO and a USB OTG bus are configured to provide MicroSD Flash storage and USB host access. With an embedded real-time memory accelerator, a multi-AHB bus matrix and two dual-port DMA controllers, a Dhrystone Performance of 1.25 DMIPS/MHz can be achieved, which is enables the design of complex embedded applications. The STM32F407 supports a maximum frequency of 168
MHz, that should be sufficient to run the envisioned traffic sensing and estimation algorithms in real time. Furthermore, this MCU has a math coprocessor (FPU) with a DSP compatible instructions set that greatly accelerate computations, in particular linear algebra operations (see Section 5).

3.2. Communications

The transmission of data between different sensor nodes requires the use of a radio transceiver. For this platform, we selected a XBEE Pro radio from Digi working at 2.4GHz, using the IEEE 802.15.4 standard. This transceiver is capable of generating signals up to +18dBm, which is the maximal legally allowed transmission power in the 2.4 GHz band in Saudi Arabia (equivalent to 100 mW EIRP when combined with a 2 dB dipole antenna). While there are a number of 802.15.4 compliant radio transceivers available, such as the TI CC2500, their maximal radiated power is insufficient for our application. The selected radio transceiver allows wireless firmware updates (e.g. OTA), which is a requirement for the project, as nodes are to be installed in unaccessible locations. We developed drivers for this transceiver to support various applications such as broadcast/unicast, packet handling, encryption, received signal strength indication (RSSI), link quality, built-in data packet building and transmission error detection. In addition, an energy-aware routing protocol is currently developed to maximize the worst case node energy under a periodic power availability pattern (which is the case for solar energy). This routing protocol is expected to be part of the middleware.

3.3. Data Storage

The MCU has an internal 1MB Flash memory for storing the bootloader, firmware, the operating system and a 196 KB internal SRAM for data during firmware execution. While this amount of memory is sufficient for all real-time processes to perform their computations, we need additional storage for non-volatile data (for instance training data used for the machine learning based tasks, historical energy data or network routing tables for the routing protocol).
We thus added a 16 KB EEPROM, a 32 MB Flash memory and a micro SD slot (Micro SD Flash Socket). The 16 KB EEPROM is connected to the MCU with I2C and stores important configuration parameters for conducting local network communication and performing OTA, such as channel, PAN ID, local network table, as well as the node configuration (i.e. the description of the set of all attached sensors). The Flash memory connects to the MCU via a SPI bus. It stores non-volatile energy and communication data, such as historical energy values, global routing tables to all possible gateways, link strength statistics (RSSI distribution), and energy statistics of other nodes (for the routing protocol). This ensures that data will remain available even if power is lost, for instance during a hardware reset. The Micro SD card socket is accessed through a SDIO interface, and supports up to 2 GB micro SD cards (FAT 16), and can be used to store the OTA firmware and to log sensors operation for debugging, or to feed the sensor fault detection algorithms.

3.4. Energy Harvesting and Storage

To make the system easier to deploy in urban environments, the platform will draw its power from a battery and an energy harvesting device. Since our system is deployed outdoors, the energy harvesting must be based on a source of energy that is readily available in cities. Many different types of energy harvesters have been proposed in the literature, for instance piezoelectric or thermoelectric generators, photovoltaic cells or wind generators. In the proposed application, we chose solar (photovoltaic) cells as they are the most reliable source of energy in urban environments \[27\]. Since solar energy is abundant, it makes sense to maximize the performance of the solar panel to maximize the efficiency of the energy harvesting, allowing a smaller solar panel to be used. The maximization of energy harvesting is based on the existence of an optimum point for the solar cell to achieve its maximal power, and this point is a function of light intensity and temperature.

To achieve this, we added to the power regulation circuitry a PWM regulator and a Maximum Power Point Tracking (MPPT) circuitry. The PWM switching
regulator is put between the solar cell and the charger chip to set the optimum input voltage for the charger chip, thus improving the charging efficiency. The MPPT circuitry is designed to track and hold the solar panel maximum power output. It uses a comparator to perform a Fractional Open-Circuit Voltage sensing [28] [29]. However, we modified the method by using an additional photovoltaic cell inside the solar panel. This is done mainly for two purposes: no external sensor is needed for this system, and the sensing photovoltaic cell will sense the same conditions as the rest of the solar panel (dust, illumination).

Energy storage is achieved by a charger chip and a battery. In order to achieve excellent high and low temperature performance and low capacity degradation over extended deployments, we selected a 8 A.h LiFePO\textsubscript{4} battery instead of Lithium-Ion or Lithium Polymer alternatives. The experience gathered during a one year deployment by our research group has shown us that Li-Ion batteries (used by the previous system) are not suitable for deployment in outdoors environments with high temperatures (between 30 to 45 degrees Celsius), in which their maximal capacity quickly degrades. Lithium polymer batteries are also not adapted to our application, as their maximal number of cycles is very limited (and do not allow extended operations during a multi-year period).

3.5. Power Management

Power consumption is a key parameter in self powered wireless sensor networks, and is even more critical in the present application where sensing and computational activities are very frequent and require a significant amount of energy [30]. The platform optimizes its power consumption through different methods, such as an energy aware routing protocol [25], an energy aware distributed computing scheme and an energy aware scheduler. In terms of hardware, the frequency of the microcontroller unit can be dynamically adjusted. Since the computational efficiency (in terms of energy per operation) is almost independent of the clock frequency, we chose a bang-bang controller that will select either the minimal operating frequency or the maximal operating frequency in function of the task to perform. The tasks are first sorted in two categories.
in function of their complexity and of the computational time horizon. For low complexity tasks such as sensing, data packet processing, received data handling or other low priority computational tasks, the microcontroller will be set to its lowest frequency. However, for high complexity tasks with strong real time constraints such as machine learning based location estimation or the mixed integer linear programming based traffic estimation, the MCU will be set to its highest frequency. More sophisticated control schemes are available, for instance [31] and [32] use a Markov Decision Process (MDP), however, these require a significant computational overhead and large data storage, and would thus offer no benefit in the present application. Indeed, the energetic cost of switching frequencies is very low, as switching requires only a few hundred clock cycles, which is comparable the number of clock cycles required for running a MDP or a more sophisticated control scheme.

At the node level, power consumption can be further minimized by switching into energy saving modes. Each node has four energy saving modes: sleep, deep sleep, hibernate and power off. Once a proper mode is selected, the microcontroller assigns control bits to the power management unit to activate/inactivate functioning devices accordingly. In normal mode, all components are powered and the platform power consumption is maximal. In sleep mode, the main MCU program is stopped, most of external components can be powered off, such as the transceiver, while the EEPROM and external flash are forced to sleep. The platform can wake-up by interrupts from the transceiver (if powered on), sensors, both RTCs or the watchdog. In deep sleep mode, the main program is stopped, most external peripherals are powered off, and most internal MCU peripherals are stopped, with the exception of the RTCs and the watchdog. In this mode, the platform can only respond to interrupts from the RTCs (internal and external) or the watchdog. In hibernate mode, all functioning components are powered off except the internal RTC and watchdog, and the MCU can only respond to interrupts from it. In this mode the total power consumption is as low as $42\mu A @ 3.3V$. 


3.6. Peripherals

The peripherals consist of several functioning blocks: a self-resetting circuitry, a battery monitoring circuitry and a USB monitoring unit.

Since no firmware is perfectly coded, we have to anticipate the presence of software bugs, which could cause normal sensing operations to fail. This is particularly important in our project, since nodes are deployed in locations that are difficult to access (for public safety reasons). Another risk lies in the remote software updates from the OTA, which could cause node failures, and accessing the failing nodes after deployment to fix them would be economically impractical. Therefore, we included a self-resetting circuitry to prevent complete node failure. The circuitry works similarly to an external watchdog, but it supports much longer timeout intervals, up to 262143 seconds (more than 1 day) in the present case. The functionality of the circuitry is to reset the whole system periodically no matter if it is functioning normally or not, and return the microcontroller control to the bootloader. The bootloader invokes an verified and trusted image. The trusted image is based on a reliable firmware which has experienced extensive testing, consisting of all necessary programs to perform sensing and communication tasks, as well as the configuration of the self-resetting circuitry. As soon as the self-resetting circuitry is configured, we can block any accidental access to it by setting a GPIO. The self-resetting can also be instantly activated using another GPIO, which can be controlled by the MCU or by the transceiver, enabling remote node hard reset (if at least the transceiver is responsive).

A battery monitor is implemented to check the battery status. The monitor consists of an energy gauge (DS2745) providing a 16 bit bi-directional current and a 11 bit voltage and temperature measurement. The current measurement and cumulated charge integration are accomplished through a voltage drop on a sensing resistance. In our design, the energy gauge and the negative temperature coefficient (NTC) resistance are soldered on a tiny board mounted on the LiFePO₄ battery, and the sensing resistance (50 mΩ) is put on the main board. A low power differential amplifier with calibrated offset and accuracy
is configured to sense the voltage drop, as shown in Figure 3. The figure also shows a LED and switch extension board which is used for extending 6 LEDs and switch on the external enclosures, and a GPIO extension board used for facilitating further development from reserved GPIO ports.

Our two years’ experience [13] with the development of a routing protocol strongly suggest that a monitoring circuity for radio data is of critical importance for debugging and transceiver fault detection. Ideally, the monitoring circuity should be able to monitor both transmission and reception simultaneously. However, most platforms developed previously did not include this functionality. Some open source platforms, such as Arduino Uno platforms or Libelium Waspmote platforms have an USB interface converted through a FTDI chip. The FTDI chip shares the same UART bus with the radio transceiver to save UART resources (there are only two UART ports for these platforms). Such configuration provides developers with a way to analyze either received or transmitted data. However, with such a configuration, it is impossible to monitor and analyze both fluxes simultaneously, which makes it extremely difficult to monitor complex networks to find hidden bugs in the API. Moreover, sharing the same UART bus with the radio transceiver poses a very important reliabil-
ity problem in practice, since the USB is also used to upload firmware. Thus, once the program jumps to the booting section, the bootloader could misread an incoming data packet (from the transceiver) as a new firmware, freezing the microcontroller in the process. The details of the the radio monitoring circuitry are shown in Figure 4. Two GPIOs are used to select three monitoring modes to analyze transmission, reception, or both.

3.7. Sensors

The sensors are deployed and installed on street lights, and consist in a lightweight (7kg) aluminum structure equipped with two infrared sensor arrays (monitoring the traffic on two lanes) and an ultrasonic rangefinder, illustrated in Figure 5. The infrared sensor arrays are used for vehicles detection and to compensate the measurements of the ultrasonic rangefinder, to detect flash flood events. Each infrared sensors array includes a set of three remote temperature sensors (Melexis MLX90614) arranged with certain angles, and targeting one traffic lane. Vehicles are detected by the small temperature variations caused by their presence. This system has a very good performance for detecting and classifying vehicles, with more than 90% accuracy [22, 34].

The ultrasonic rangefinder (Maxbotix MB7076) measures the distance between the sensor and ground level with a resolution of 1 cm, without temperature compensation.
Figure 5: **Passive IR/ultrasonic traffic sensor used for the study.** This figure shows an actual node deployed on KAUST campus. The two passive infrared sensor arrays are mounted at different angles and positions, to monitor two lanes of traffic simultaneously. The ultrasonic rangefinder is mounted perpendicularly to the surfaces to monitor (road and vehicles) to maximize the return signal amplitude.

This traffic sensor is also designed to monitor water levels in streets, which allows the complete system to detect flash floods in cities (this project was undertaken following the 2009-2011 flash floods events in Jeddah that caused hundreds of casualties in the city). The presence of water is detected by monitoring the ground temperature, which severely drops whenever a flood occurs. In addition, the water level is measured using the ultrasonic rangefinder, and air layer temperature estimation based on the remote infrared sensors. Indeed, the **Melexis MLX90614** measures both the ground temperature and the local temperature of the sensor, and allows the compensation of the variations in speed of sound caused by temperature inhomogeneities. By using Artificial Neural Networks (which we outline in the subsequent sections), a 1 cm accuracy can be achieved, which corresponds to less than 0.2% of relative error.

4. **Software**

This platform runs **UCOSII**, a priority based preemptive real-time multitasking operating system kernel for microprocessors. It is highly portable, and manages up to 64 tasks, only requires a few kilobytes of code and a few hundred bytes of RAM to provide multitasking scheduling, memory management, interrupt handling, synchronization primitives, etc. The source code is mostly written in ANSI C, thus developers can write the application code in ANSI,
avoiding the need to learn a specialized language (such as NesC for TinyOS). Contrary to other embedded OS for WSN, which are based on event-driven mechanisms, uCOSII outperforms in terms of real-time performance due to the preemptive, priority-based task scheduling mechanism. In contrast, the event-driven OS, such as TinyOS or Contiki, may require less RAM and ROM space, but strictly are not real time OS. For example, TinyOS is based on a First In, First Out (FIFO) scheduling policy, while Contiki uses a polling mechanism. For our application, both the traffic monitoring/control and flash flood detection requires certain level of real time, thus the event driven OS is not a competent selection for us. As shown in figure 6, the OS is ported to the hardware with three files and the user application library is developed by ANSI C or C++.

Programmers can build middleware layer based on the OS to provide services upon their requests, and this allows WSN developers to integrate operating systems and hardware with the wide variety of various applications that are currently available [35]. Moreover, the use of middleware considerably enhances the code portability. Currently, a middleware is being developed to on-line estimate the power harvesting and consumption pattern with ambient variation, which will facilitate a more efficient energy management on both mote and network levels. Also, the developing middleware will be able to perform a lightweight online fault-diagnose by monitoring the energy, data transfer and execution, and reset the module or the system. This will be critical to ensure that nodes do not fail due to errors, and that the computational methods used on this platform are properly executed.

The 1Mbytes embedded program flash is divided into three sections: main program, boot section and ota section, as shown in Figure 7. The left figure shows the program memory allocation, the first 128KB Flash space is configured as boot section, in which it includes the firmware for initializing and testing the hardware components, and signaling (by pre-defined LED blinking pattern) to the user the status. It is the first firmware executed after the microcontroller powers on, resets, or reboots. Also, the firmware consists of the codes to support updating main program firmware by USB-OTG port. Upon the completion of
Figure 6: Software architecture

Figure 7: Left: Embedded program memory configuration; Right: Operation mode

the boot firmware, the program pointer jumps to the entry address of the main program section. The 128KB OTA section is configured for OTA (over-the-air) application, which enables to update the main program by wireless. The main program jumps to this section once any OTA command is received, and then starts to save the received firmware in SD. Upon the checksum getting validated and firmware execution command is confirmed, the new firmware is loaded into the main program. In order to avoid the main program deadlock, a validated and trusted firmware can be put in the SD before the deployment. The boot firmware will load the trusted firmware if a bad firmware stuck the program.

In order to achieve a tradeoff between power consumption and computational capability, the clock frequency is adjusted online. For example, the following
functions can be used to configure a task with a system clock of 32Mhz.

SysClkWasp=32000000; //System clock switch to 32MHz.
SysPreparePara(); //System parameter preparation
SystemInit(); //System initialization

By this way, the system clock can be adjustable to satisfy the request of each specific application and task.

5. Platform evaluation and example applications

5.1. Cost Evaluation

Table 1 lists the costs and functions of some major components used for the proposed platform. The entire platform cost is around 110 dollars excluding sensors for small quantity (less than 20) manufacturing. For mass production, the cost can be further reduced to less than 80 dollars once the quantity exceeds the break down quantity (normally 1000). The cost of the sensors depends on the number of traffic lanes, for a two lanes road, the total price is around 235 dollar. Similarly, the cost of the sensors will be decreased with mass production.

5.2. Power consumption

Power consumption is a critical issue for solar/battery powered wireless sensor nodes. To assess it, it is necessary to determine the contribution of each module to the entire power consumption. A series of tests have been conducted to evaluate the power consumption of the platform investigated in this article. Table 2 summarizes the contribution of major components in different operating modes, all components working at 3.3V. Owing to the dynamic adjustment of the clock frequency, the MCU is able to switch between low power sensing (4.1mA @ 3.3V @ 8MHz) to high power computation (87.0mA @ 3.3V @ 168MHz), thus achieving a tradeoff between power and pure performance. Out of all power saving modes, the hibernate mode has the lowest power consumption. Note that the radio transceiver is a major power consumer: with the
Table 1: Cost of the major components

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Price $</th>
<th>Breakdown price $</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F407</td>
<td>1</td>
<td>11.06</td>
<td>7.18@1000</td>
<td>Micro-controller</td>
</tr>
<tr>
<td>MP2635</td>
<td>1</td>
<td>3.98</td>
<td>2.29@500</td>
<td>Power Management</td>
</tr>
<tr>
<td>DS3231</td>
<td>1</td>
<td>7.72</td>
<td>4.5@1000</td>
<td>Real Time Counter</td>
</tr>
<tr>
<td>FT232RL</td>
<td>1</td>
<td>3.11</td>
<td>2.75@2000</td>
<td>RS232-USB Converter</td>
</tr>
<tr>
<td>BR24T128F</td>
<td>1</td>
<td>0.66</td>
<td>0.34@2000</td>
<td>32Kbits EEPROM</td>
</tr>
<tr>
<td>W25Q256F</td>
<td>1</td>
<td>2.43</td>
<td>1.99@1000</td>
<td>256MBits Flash</td>
</tr>
<tr>
<td>DS2745</td>
<td>1</td>
<td>4.62</td>
<td>2.09@3000</td>
<td>Battery Monitor</td>
</tr>
<tr>
<td>NCP603</td>
<td>6</td>
<td>0.57</td>
<td>0.19@3000</td>
<td>Voltage Regulator</td>
</tr>
<tr>
<td>TI 74CB3Q</td>
<td>1</td>
<td>0.95</td>
<td>0.39@2000</td>
<td>Quad Channel MUX</td>
</tr>
<tr>
<td>XBEE PRO</td>
<td>1</td>
<td>32</td>
<td>32</td>
<td>2.4Ghz Transceiver</td>
</tr>
<tr>
<td>MB7076</td>
<td>1</td>
<td>114.95</td>
<td>72.5@100</td>
<td>Ultrasound Sensor</td>
</tr>
<tr>
<td>MLX90614</td>
<td>6</td>
<td>20.4</td>
<td>14.1@500</td>
<td>Infrared Sensor</td>
</tr>
</tbody>
</table>

highest authorized RF power output (+18dBm), the radio module has the highest continuous power consumption (250mA@3.3V). Power consumption is still high in the listening mode, with a consumption of 55mA@3.3V. Figure 8 illustrates the battery discharging and charging behavior of one node from 18:00pm to the next day 21:00pm based on multiple days’ measurement. The positive value of battery current represents discharging (without harvesting), and the negative value indicates the charging behavior due to the solar harvesting. It can be seen that the charging and discharging current achieve an equilibrium (zero value) around 9:00 am in the morning, and the solar energy harvesting approaches the maximum around 11:00am.

In order to estimate the typical battery life during sensing operations, we define two benchmark sensing conditions: the heavy duty condition represents the case in which sensing, transmission and computational power are the highest. We define it as using the transceiver in listening mode for 50% of the time, with computations at maximal clock frequency during 40% of the time, the remainder being transmissions at maximal RF power. The normal duty stands for a more typical sensing operation in which using the transceiver in listening mode for 20% of the time, with computations at maximal clock frequency during 20% of
Table 2: Power consumption of the individual components.

<table>
<thead>
<tr>
<th>Device</th>
<th>Mode</th>
<th>Operation</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCU</td>
<td>Active</td>
<td>Sensing@8MHz</td>
<td>13.5 mW</td>
</tr>
<tr>
<td></td>
<td>Active</td>
<td>Computation @168MHz</td>
<td>287.1 mW</td>
</tr>
<tr>
<td></td>
<td>Sleep</td>
<td>Peripherals Enabled</td>
<td>6.6 mW</td>
</tr>
<tr>
<td></td>
<td>Deep Sleep</td>
<td></td>
<td>1.2 mW</td>
</tr>
<tr>
<td></td>
<td>Hibernate</td>
<td></td>
<td>5.6 µW</td>
</tr>
<tr>
<td>Radio</td>
<td>TX</td>
<td>+18dbm, 250kbps</td>
<td>825.0 mW</td>
</tr>
<tr>
<td></td>
<td>TX</td>
<td>0dBm, 250kbps</td>
<td>148.5 mW</td>
</tr>
<tr>
<td></td>
<td>RX/Idle</td>
<td></td>
<td>181.5 mW</td>
</tr>
<tr>
<td>Flash</td>
<td>Active</td>
<td>Read Data</td>
<td>13.2 mW</td>
</tr>
<tr>
<td></td>
<td>Active</td>
<td>Write Data</td>
<td>26.4 mW</td>
</tr>
<tr>
<td></td>
<td>Standby</td>
<td></td>
<td>33.0 µW</td>
</tr>
<tr>
<td>EEPROM</td>
<td>Active</td>
<td>Write/Read</td>
<td>8.3 mW</td>
</tr>
<tr>
<td></td>
<td>Standby</td>
<td></td>
<td>0.1 mW</td>
</tr>
<tr>
<td>RTC</td>
<td>Active</td>
<td></td>
<td>0.7 mW</td>
</tr>
</tbody>
</table>

Figure 8: Battery charging and discharging over multiple days
Table 3: Battery life estimation in different modes of operation.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Avg. Power</th>
<th>Battery Life</th>
<th>Interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heavy Duty</td>
<td>330.0 mW</td>
<td>3.3 Days</td>
<td>All</td>
</tr>
<tr>
<td>Normal Duty</td>
<td>221.1 mW</td>
<td>5.0 Days</td>
<td>All</td>
</tr>
<tr>
<td>Sensing</td>
<td>46.2 mW</td>
<td>23.9 Days</td>
<td>All excluding Xbee</td>
</tr>
<tr>
<td>Sleep</td>
<td>6.6 mW</td>
<td>163 Days</td>
<td>All IO interrupts, Int. and Ext.RTC, watchdog</td>
</tr>
<tr>
<td>Deep Sleep</td>
<td>1.3 mW</td>
<td>854 Days</td>
<td>Int. and Ext. RTC, watchdog</td>
</tr>
<tr>
<td>Hibernate</td>
<td>132.0 µW</td>
<td>7993 Days</td>
<td>Int. RTC, watchdog</td>
</tr>
</tbody>
</table>

the time, and transmissions during 10% of the time. Table 3 shows the battery life with different operating modes, with the assumption that no solar energy is available. Using these benchmarks, the node can be operational for 3 days and 5 days in heavy duty and normal duty modes respectively. If the platform is only used as a sensing and data logger with minimal data exchange and computation, a continuous operation of 24 days can be obtained. The platform achieves the longest operation time in hibernate mode (theoretically more than 20 years, though the shelf life of the battery will be a limiting factor).

5.3. Benchmark computations

Since this platform will be the backbone of the proposed traffic and flash flood monitoring system, it needs to be fast enough to execute complex calculations. We implemented some benchmark tests to evaluate the computation capability of the new hardware platform, using various unit tests on matrix operations. In our tests, we used a CMSIS-compliant DSP library and an open-source matrix library. Figure 9 and Figure 10 illustrate the computational times for different dimension matrix additions and multiplications using either the CMSIS library or the C library.

These two benchmarks (matrix addition and multiplication) are very important in practice, since most processing operations require a combination of these two operations. For example, the Kalman Filter, commonly used to estimate
the state of a linear system heavily uses matrix operations for propagating and updating the state.

In the present application, the nodes have to estimate the state of traffic in real time, which can be achieved for example using an Ensemble Kalman Filter or a Particle Filter, as in [36], or using an optimization formulation as in [37], [38]. All of these approaches require computationally intensive matrix operations (addition, multiplication, and inversion). Similarly, the training and prediction by Artificial Neural Networks (which we outline later) require both matrix multiplications and additions.

The main difference between the two is that the CMSIS library makes the use of the STM32F407 internal hardware accelerator (FPU) and the associated DSP library.

In Figure 9, 6 different matrix dimensions (5×5, 10×10, 20×20, 30×30, 40×40, 50×50) were tested for addition. As expected, we can see that the computational time obtained using the CMSIS library is less than the one based on the C library. For a 10×10 matrix addition, the time required by the CMSIS library is 33 times faster than the same computation by the C/C++ library, while for
Figure 10: Computational time of a benchmark matrix multiplication problem.
This figure compares the CMSIS (DSP) library with the C library.

For a 50x50 matrix addition it is 57 times faster.

We have similar results for matrix multiplication, as shown in Figure 10.

We also evaluate an Extended Kalman Filter (EKF) benchmark at different operating frequencies based on CMSIS-DSP library and an open-source EKF library (TinyEKF [39]) in C/C++ implementation. The benchmark EKF has an 8x8 state matrix and a 4x8 measurement matrix. Following Table 4 summarizes a performance comparison regarding execution time, power consumption and energy of one iteration of the EKF computation. It is obvious that the use of hardware accelerator (FPU) significantly reduces the execution time, while the additional power consumption due to the hardware accelerator is limited. As a result, the energy consumption of FPU-based execution is more than 10 times less than that of C/C++ based implementation. More importantly, the computation energy and efficiency improves with the increase of clock frequency. A higher clock frequency can lead to a lower energy consumption and better efficiency. Thanks to our on-line clock adjustment firmware, the proposed platform can self-increase the clock to minimize the energy cost and better the efficiency during computation, thus achieving a good tradeoff between high load computa-
Table 4: Performance comparisons of EKF with/without FPU acceleration.

<table>
<thead>
<tr>
<th></th>
<th>32 Mhz</th>
<th>64 Mhz</th>
<th>96 Mhz</th>
<th>128 Mhz</th>
<th>160 Mhz</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Execution Time (µs)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>With FPU Acceleration</td>
<td>745</td>
<td>375</td>
<td>252</td>
<td>191</td>
<td>155</td>
</tr>
<tr>
<td>Without FPU Acceleration</td>
<td>8670</td>
<td>4340</td>
<td>2900</td>
<td>2180</td>
<td>1740</td>
</tr>
<tr>
<td><strong>Power Consumption (mW)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>With FPU Acceleration</td>
<td>129.36</td>
<td>154.11</td>
<td>178.86</td>
<td>203.61</td>
<td>228.03</td>
</tr>
<tr>
<td>Without FPU Acceleration</td>
<td>128.37</td>
<td>152.79</td>
<td>177.87</td>
<td>202.62</td>
<td>227.04</td>
</tr>
<tr>
<td><strong>Energy Consumption (µJ )</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>With FPU Acceleration</td>
<td>96.37</td>
<td>57.79</td>
<td>45.13</td>
<td>38.95</td>
<td>35.23</td>
</tr>
<tr>
<td>Without FPU Acceleration</td>
<td>1112.97</td>
<td>663.11</td>
<td>514.93</td>
<td>440.70</td>
<td>395.05</td>
</tr>
</tbody>
</table>

... (rest of the text continues with detailed explanations and applications)
and air temperature inputs measured by the passive infrared sensors. The ANN is part of a supervised learning approach, in which the supervisory signal is the compensation to apply to the raw ultrasonic distance measurements, which can be inferred from the ultrasound distance signal generated by the sensor, assuming that no flood currently occurs. During testing [41], we found that naive compensation based on simple atmospheric models were inadequate, and resulted in errors on the order of tens of centimeter, more than the expected water level of a small flash flood.

5.4.1. Implementation of a supervised learning algorithm on the proposed computational platform

While efficient, this ANN approach however requires significant embedded computational power. We now outline the implementation of an ANN in this computational platform.

5.4.2. Neural Network Training

In this application, we consider a Levenberg-Marquardt training function. Training a neural network involves the tuning of the weights and biases of the network. The objective is to maximize the network prediction performance, which corresponds to minimizing the difference between all network outputs $y_k$ and desired outputs or targets $t_k$ on validation data. The computational time required for the training algorithm depends on many factors, including the complexity of the problem, the number of data points in the training set, the number of weights and biases in the network, the norm function used as the objective function of the problem, and whether the network is being used for pattern recognition (discriminant analysis) or function approximation (regression). For our particular problem, we are interested in the function approximation problem with a few hundred weights in a moderate size network. In this specific case, the Levenberg-Marquardt algorithm has been proven to have the fastest convergence [42]. It updates the network weights and biases in the direction in which the performance function decreases most rapidly. This advantage is more
noticeable if a very accurate training is required.

The classical norm used for the training and validation of the feedforward function is the $L_2$ norm \cite{43}, which amounts to the mean squared error (MSE) between the network outputs and the target outputs. Given a training set including a set of input vectors $\{x_n\}$, where $n = 1, ..., N, x_n \in \mathbb{R}^D$, together with the corresponding set of target vectors $\{t_n\} t_n \in \mathbb{R}^k$, our objective is to minimize the error function (in the context of the Levenberg-Marquardt algorithm) in the $L_2$ norm sense:

$$E(w) = \frac{1}{N} \sum_{n=1}^{N} \|t_n - y(x_n, w)\|^2$$  \hspace{1cm} (1)

Similarly to other numerical minimization schemes, the Levenberg-Marquardt algorithm is an iterative procedure. It is initialized with a given parameter vector, $\{w\}$. During each iteration step, the parameter vector, $w$, is replaced by a new estimate, $w + \delta$. To determine $\delta$, the functions $y(x_n, w + \delta)$ are approximated by their linearizations:

$$y(x_n, w + \delta) \approx y(x_n, w) + J_n \delta$$

where $J_n = \frac{\partial y(x_n, w)}{\partial w}$ is the gradient of $y$ with respect to $w$. At the minimum of the sum of squares $E(w)$ in Equation 1, the gradient of $E$ with respect to $\delta$ is zero. The above first-order approximation of $y(x_n, w + \delta)$ gives

$$E(w + \delta) \approx \sum_{n=1}^{N} (t_n - y(x_n, w) - J_n \delta)^2$$

which can be rewritten as:

$$E(w + \delta) \approx \|t - y(w) - J \delta\|^2.$$  \hspace{1cm} (2)

Taking the derivative with respect to $\delta$ and setting the result to zero yields

$$(J^T J) \delta = J^T [t - y(w)]$$

where $J$ is the Jacobian matrix whose $n$th row equals $J_n$, and where $y$ and $t$ are vectors with $n$th component $y(x_n, w)$ and $t_n$, respectively. This is a set of linear
equations which can be solved for $\delta$. With minor modifications (through the addition of a damping term $\lambda$), we obtain the Levenberg-Marquardt algorithm:

$$(J^T J + \lambda \text{diag}(J^T J)) \delta = J^T [t - y(w)].$$  

(2)

5.4.3. Implementation

The implementation of code is done using Keil v4.7 from the ARM group, and optimized for C/C++. We have implemented our algorithms on the wireless sensor nodes using a conventional back-propagation neural network class in C language that makes use of gradient descent, with parameters defined as: 0.001 learning rate, 1500 of maximum epochs during training, and maximum accuracy. The demonstration code is a 996 lines written in C++ language and is built on top of <math.h>, <algorithm>, <fstream>, <string>, <vector>, <stdio.h> libraries, as well as the designed neural network class "neuralnetwork.h". The code can toggle between batch and online training mode, and gets the training stopping criteria from the user. Its total memory size (when compiled) is 101 kB, while its peak memory usage is 58 kB, which is within the limits of the microcontroller.

On the platform investigated in this article, the online training mode takes about 2 hours to process one week of temperature and distance measurement data (sampled at 10 Hz), while the prediction is quite fast 0.03 seconds per data sample, which is slightly faster than real time. The residual error in water level estimation (using online training of the ANN) is shown in Figure 11.

Since the ANN captures the compensation and apply very accurately, no false detection has been observed during a 12 month test period. A minor flash flood event was detected using two of these sensors, deployed in Umm Al Qura University (Mecca, Western Saudi Arabia) in 2014 [40], as can be seen in Figure 12. The estimated water level during the floods were consistent with observations that the flood was minor, and that the level was under 10 cm according to local record.
5.4.4. Energy analysis

In the flash flood monitoring application, it is critical to train the ANN parameters in each sensor. Indeed, sending one week of temperature and distance measurement data to a computer would require approximately 70MB of data (data sampled at 10 Hz, with six parameters and two bytes per parameter). The XBee radio has a theoretical maximal output of 30kB/s, though it is closer to 3 kB/s in real life situations (sensors separated by 100 m), as reported in [13]. Therefore, transferring all this data to the sensor network gateway would require approximately 7 hours per sensor, and considerably more if data requires more than one hop to reach the gateway. In addition to the considerable bandwidth usage (which would affect the communication with other sensors), this would lead to a prohibitive energy consumption, on the order of the total capacity of the battery (assuming 7 hours and 400mA average consumption), which would cause the inner nodes (closest to the gateway) to run out of energy since they would have to relay data generated by other nodes connecting to the gateway through them.
As shown in the previous subsection, the platform developed in this article can perform these computations onboard in about two hours, using a fraction of the energy required to send this data to the gateway.

6. Conclusion

This article presents the design and implementation of a new computational platform that can enable complex smart city applications. The primary motivation for this design was the lack of a suitable commercially-available computational platform that is adapted to extended deployments in urban environments. The platform design files have been uploaded and shared in open science framework, and can be accessed from [1]. The hardware design is under CERN Open Hardware License v1.2.

The main focus of the present article is flash flood monitoring using a combination of ultrasonic and infrared temperature data, processed using Artificial Neural Networks. The present platform has also been explored for other applications, such as in [25-34].
This platform can enable sensor networks that rely heavily on computing for their operation, and for which range and energy supply are important constraints. Such sensor networks, are much easier to deploy because they do not require backend computational equipment, and they are more energy efficient (allowing smaller batteries and solar panels to be used), and occupy less bandwidth. While the present article focuses on flash flood monitoring, these sensor networks are particularly adapted to traffic monitoring, in which we are not interested in keeping all intermediate sensing data that are used for estimating the state of traffic. Future work will be dedicated to allowing ultra-low power operation by lowering the microcontroller voltage under very low power setting. The programming of a dedicated middleware adapted to advanced smart cities operations will also be critical to allow reliable operation for extended periods of time, while managing energy, real time and network communication constraints efficiently.

References

References


