2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), and 2015 IEEE 12th International Conf on Embedded Software and Systems (ICESS)

Sensor-Based Low Power Management For Mobile Platforms

Douglas Lautner[‡], Scott DeBates[†], Jagat Shah[†], Miao Song[‡], Shangping Ren[‡]
[‡]Illinois Institute of Technology, Chicago, IL 60616 USA
[†] Motorola Mobility LLC., Chicago, IL 60654 USA
Email: dlautner@hawk.iit.edu, {scottdebates, jagat}@motorola.com, msong8@hawk.iit.edu, ren@iit.edu

***Abstract*— As mobile devices become increasingly more advanced and essential for everyday life, minimizing power consumption of these devices has also become one of the critical design concerns, especially for those mobile devices that support "always-on" applications, such as a driving mode detection application. In this paper, we present a design, implementation, and deployment of an energy efficient mobile platform for end-consumers. Taking the driving mode detection application as an example, we give the platform design and technical implementation details. Our solution not only offers a clean and portable design for all sensor-related applications, it is also proven that it can significantly reduce power consumption. In fact, based on live measurements, for driving detection applications, the designed platform provides up to 73.11% energy consumption reduction, and up to 18.43 minutes of prolonged battery life. To the best of our knowledge, the platform is the first solution to provide a clean, portable, and fundamentally energy efficient architecture for Andriod sensor related applications. It has been deployed in Motorola commercial devices.**

***Index Terms*— Android, embedded system, energy efficient, low power, mobile device, mobile platform, sensor fusion core**

I. Introduction

Advanced mobile devices such as smart phones, tablets, and wearables are playing a large role in improving the quality of our daily lives. Humans use such devices to communicate with the environment and each other through different sensing capacities. For instance, current sensing capabilities on mobile phones include WiFi, Bluetooth, GPS, audio, video, light sensors, accelerometers, etc. In other words, mobile devices are "no longer only a communication device, but also a powerful environmental sensing unit that can monitor a user's ambient context, both unobtrusively and in real time"[1]. More and more applications are being built on mobile devices that exploit such sensing services providing context-aware functionality benefiting end-users in their daily lives. A new example is drive mode detection. While driving, it is normally not convenient or even forbidden for users to text back or call back when a new message or call comes. By detecting whether the user is in drive mode or not, an application can assistant the user depending on the contextual situations. For instance, the application can forward the users call or automatically reply to the message or voice-play the message for users when they are in the drive mode.

However, these applications can quickly drain the limited battery of mobile devices using sensing technologies. Sensors are famously known as big energy consumption sources, especially those that are frequently triggered to collect raw data. By interrupting the application processor in mobile device to process raw data in a high frequency, these sensors and the related applications will cost a large amount of current drain. How to prolong the life of battery, yet achieve the same functionality and better energy efficiency of these applications, becomes a big challenge and the focus of this paper.

Oftentimes, for an application to sense and obtain the information of its surroundings, a direct approach is to employ and exploit the related sensing capabilities. In the drive mode detection example, GPS signals are normally continuously requested by applications to detect the speed/location of a vehicle. Based on that, the applications can easily calculate the conclusion. Effective as it may be, this approach is very energy costing. A careful study indicates that some features of drive mode can be detected or identified by sensors, such as the accelerometer, are much more energy efficient. By the coordination of various sensors data and GPS signals, we have proven an energy efficient solution without comprising the functionality of the applications.

In particular, the solution should support: 1) various sensor data and GPS signals need to be seamlessly and conveniently coordinated in the mobile device; 2) the use of sensor data itself should not cost too much current drain; and 3) no constraints to only support drive mode detection. As a matter of fact, many applications share the same characteristics with the drive mode detection. These applications involve multiple sensors and different sensing technologies. Usually they have intensive interactions with the users, which make them subject to significant amount of current drain discharge. Therefore, the proposed solution should be generic enough to fundamentally support a category of such applications.

Driven by these targets, in this paper we propose, implement and deploy to end-consumers an energy efficient mobile architecture/platform. Our formulation uses an Android operating system to support the applications with lower energy cost. We design and implement various software modules within a sensor fusion core to exploit the low energy cost capability and provide fundamental support for upper layer applications in Android's stack. Our architecture is designed to be mobile platform agnostic and not built targeting for one particular application or device. The design of our modules enables the sensor fusion core to seamlessly communicate with

978-1-4799-8937-9/15 $31.00 © 2015 IEEE
DOI 10.1109/HPCC-CSS-ICESS.2015.281

the existing sensors or wireless sensing technologies in the Android operating system. In fact, most Android sensor-based applications are able to gain an improved power saving performance without changing the current Android framework or application itself. Our solution is proven to have significantly reduced power consumption, i.e., up to 73.11% improvement in energy consumption improvement using live measurements and up to 18.43 minutes of prolonged battery life in drive mode detection. It has been deployed, measured and proven in commercial devices.

To the best of our knowledge, our platform is the first solution to provide a clean, portable and fundamental energy efficient architecture for Android sensor related applications.

We will discuss the related work in Section II, the detailed design and implementations are discussed in Section III. We verify the performance of platform implementation in Section IV. Finally, we conclude in Section V.

II. Related Work

With the big obstacle of energy consumption in sensing capabilities, much research has been devoted to investigating the efficient power management design at the application level. Various strategies are employed to reduce the unnecessary sensing frequency without compromising the functionality of the applications. For example, Wang et.al [1] proposes a framework where energy efficiency is achieved by managing sensors in a hierarchical way based on the user's current state. The core component called " Energy Efficient Mobile Sensing System (EEMSS)" associates user states with particular sensors. It contains both the set of necessary sensors needed to be monitored and the sequence of future sensors to be turned on to detect state transition. By carefully selecting a subset of sensors based on the users state, [1] can avoid unnecessary energy waste without compromising the accuracy of state detection by multiple sensors. Similarly, Zhuang et.al [2] utilizes the low accuracy requirement of the application, such as twitter, to reduce the frequency of GPS sensing. In addition, [2] uses piggybacking of multiple sensors data to reduce the total times of collecting data. Abdesslem et.al [3] comes up with the approach of switching between GPS and WiFi based location sensing. The authors keep GPS on at all times to detect if the device is indoors/outdoors; when a GPS signal is detected, the device is considered outdoors and location sensing by WiFi is turned off. Pendao et. al [4] explores periodic sampling of the sensors and the suspension of the sampling process in the Android operating system whenever the device is not moving. By balancing between sensing tasks effectiveness and the energy consumption in the context of human motion analysis, [4] can save energy without compromising the collected data through suspending the sampling process during periods of immobility. Also based on human activity sensing, Li et. al [5] presents an unobtrusive, energy-efficient approach through the intelligent scheduling of built-in sensors on mobile phones and light-weight compressed sensing. In their approach, an activity pattern matrix was constructed and adaptively modified to control the scheduling of active sensing in a Compressed Sensing (CS) mechanism. CS can effectively reduce the samples required to save energy.

Recently, there is a trend of collective mobile sensing [6] where a coordinator controls sensing activities of users such that those mobile phones sense collaboratively to produce just enough data reports for the application. Sheng et. al [7] models the energy-efficient collaborative sensing with mobile phones as the optimization problem. They propose the practical and effective heuristic algorithms to find energy-efficient sensing schedules under realistic assumptions. Their work is acclaimed to be the first to present theoretically well-founded and practically useful algorithms to show the energy-saving benefits of using collaborative sensing in mobile phone sensing applications.

Meanwhile, some researchers have started to look into the low level design: they focus on the low power embedded mobile platform design for energy saving purposes. In the Viredaz et. al's work [8] , the system hardware is suggested to be designed as a collection of inter-connected building blocks that could function independently to enable independent power management. Pillai et. al [9] introduces the dynamic frequency/voltage scaling to reduce power consumption by configuring the processor based on the requirements of the executing applications. However, these approaches do not provide a direct solution to sensing related processing. Priyantha et. al [10] presents a sensing architecture to include a dedicated low-power sensing processor for sampling and low-level processing of sensor data. However, an application using this architecture must itself decide how to partition an application across main processor and sensing processor.

In this paper, we exploit the characteristics of the sensor fusion core and present a power efficient mobile platform based on Android system especially targeting sensor-related processing. In particular, we use in-car driving mode detection application as an example to illustrate our design. However, it is worth noting that any sensor-based Android application that fits the abstraction of our design model can benefit from our software infrastructure [11], [12].

III. Energy-efficient Platform Design and Implementation for Mobile Devices

A. Approach Evolution

We take an in-car driving mode detection as an example. For a drive mode application, GPS signal is normally the intuitive and direct approach to resort to. Usually, the application continuously uses GPS scanning at the application processor to keep tracking the current location of cars, which will help to detect if the car is in the drive mode or not. Fig. 1 depicts the approach that obtains the required accuracy and detection of in-car state, however is highly energy consuming.

Our research shows that GPS scanning is not always necessary during the drive mode. For example, if the car is stopped at traffic jam or red light, the continuous GPS scanning will be a wasted energy cost as it will not provide any updated location/speed information. Under such scenarios, the GPS scanning can be turned off at the application processor.

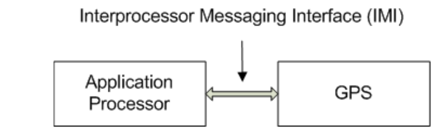


Fig. 1. The Continuous GPS Scanning Approach

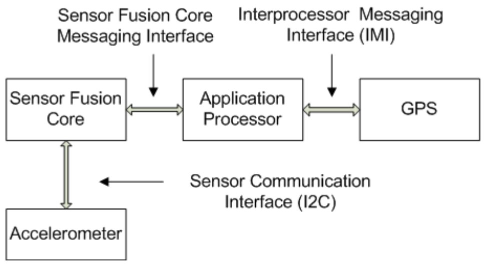

Sensor Fusion Core Messaging Interface
Interprocessor Messaging Interface (IMI)
Sensor Fusion Core
Application Processor
GPS
Sensor Communication Interface (I2C)
Accelerometer

Fig. 2. The Sensors+ Sensor Fusion Core Approach

With the help of sensors, we are able to employ the movement and non-movement detection which provides information to the application processor to start and stop GPS scanning based on movement of a device. As sensors could also be the source of high current drain at the application processor, the movement and non-movement detection algorithm needs to be refactored at the sensor fusion core level to fully exploit its energy saving features. Fig. 2 depicts the architecture of using sensor fusion core and detection refactorization.

Architecting sensor-related movement detection in the sensor fusion core and being able to communicate and coordinate with the application processor seamlessly at the same time in the Android system is a novel approach. In order to fundamentally support this feature, we provide and implement the mobile platform to fully exploit the sensor fusion core.

The specific hardware configuration of our platform is shown in Fig. 3, and we will extend the discussions of our platform in following sections.

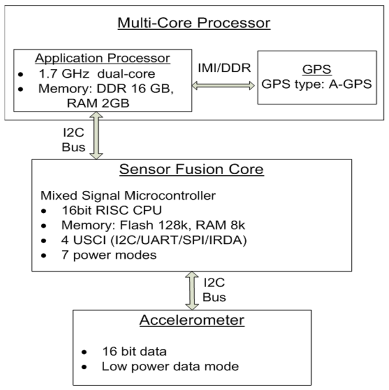


Fig. 3. Hardware Configuration of Energy-efficient Mobile Platform

B. Design of Energy-efficient Mobile Platform

Our mobile platform is designed to exploit the energy-saving feature of a sensor fusion core. How to customize the usage of a sensor fusion core is a key design of our platform. We first briefly introduce sensor fusion core's related features.

1) Sensor Fusion Core: Sensor fusion core is a flash programmable microcontroller and integrated sensors tightly coupled in the SoC (System on Chip) architecture specially designed for sensor data processing. It contains precisely defined processing power and memory to gather and process data from various sensor types. For example, our trial uses an 8KB RAM, 128 KB Flash and maximum clock speed of 25MHz. The processing capacity of the sensor fusion core makes itself a perfect solution to offload the sensor processing work from a highly power intensive processing unit, i.e., application processor.

On the other hand, the sensor fusion core can also deliver the results with low power consumption due to its special design. A sensor fusion core has three configurable power modes: *active* mode, *standby* mode, and *off* mode. When it is in *active* mode, the CPU and all clocks are turned on. When the sensor fusion core enters *off* mode, the CPU is disabled, clocks are stopped, and data is in retention status. The lowest power mode available that still supports timers is *standby* mode.

Because of its outstanding energy efficiency performance, our platform is built upon a sensor fusion core. We will discuss our architecture design within a sensor fusion core in the following sections.

2) Software Architecture within Sensor Fusion Core: In particular, the software architecture of our platform is shown in Fig. 4:

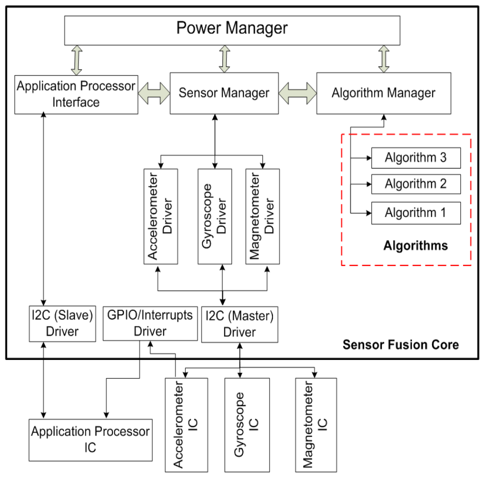


Fig. 4. Sensor Fusion Core Software Architecture

There are four key modules in the design. In particular,

- Power Manager: manages the power mode votes from different tasks and configures the sensor fusion core's power mode based on these votes.
- Sensor Manager: configures the sensors and manages the raw sensor data obtained from the sensors.
- Algorithm Manager: configures, schedules, and provides the raw sensor data for all algorithms.
- Application Processor Interface: responsible for all communications between the sensor fusion core and applica-

tion processor. Its tasks include managing features turned on/off by the application processor, and interrupting the application processor when sensor data is available to be processed.

The four modules serve different functionalities and interact with each other. Algorithm Manager manages all algorithms that are currently needed and it relies on Sensor Manager to turn on and turn off different sensors or run them at different speeds. Sensor fusion can be achieved by running multiple algorithms at the same time with various sensor data (accelerometer data only vs. accelerometer and magnetometer data) provided at different speed (50Hz vs. 100 Hz). Power Manager provides interfaces to voting mechanism that is used by all other components.

3) Interaction with Android Operating System: Our mobile platform is not built targeting for one particular application or device. Sensor fusion core itself should be able to seamlessly communicate with the other existing sensors or sensing technologies in the Android system. In other words, sensor fusion core should be treated as a part of the Android system, and the sensor related applications are abstracted and not aware of the existence of sensor fusion core with our configuration.

In our platform, sensor fusion core is treated as an individual virtual sensor to the Android system. We add three modules, i.e., sensor fusion core driver, virtual sensor Hardware Abstraction Layer (HAL) , and sensor/virtual sensor service, to bridge the sensor fusion core with the Sensor Manager in Android framework.

In particular, virtual sensors are part of the Android system that allow mobile device manufacturers the ability to create new sensors that Android has not defined. The virtual sensors normally provide Android applications the calculated sensor fusion, or contextual data. In the drive mode detection example, contextual data is provided through virtual sensor to let the application know when the users are or are not driving. All virtual sensors a manufacture supports on a device are managed by the virtual sensor HAL.

It is worth mentioning that our modules do not affect the functions of the existing Android modules, and can be removed and changed per the users needs. The position of our platform in Android system is shown in Fig. 5.

C. Key Implementations of Energy-efficient Mobile Platform

In terms of our platform implementation, there are several key implementations that are tailored to be energy-efficient and platform portable.

1) Sensor Data Interruption Mechanism: Within the sensor fusion core, the normal interruption mechanism, i.e., timers, will require the sensor fusion core to run a clock all the time. Therefore, the sensor fusion core will keep awake without a chance to enter into the *off* mode. To save 1uA under the *off* mode, we need to change the design of sensor data interruption in sensor fusion core. In particular, we do not use a timer to interrupt sensor fusion core whenever raw sensor data is available. Instead, an external GPIO interrupt is employed to wake sensor fusion core. By architecting the software system to not use timers to wake up and read sensor data, we are able to save an additional 1uA.

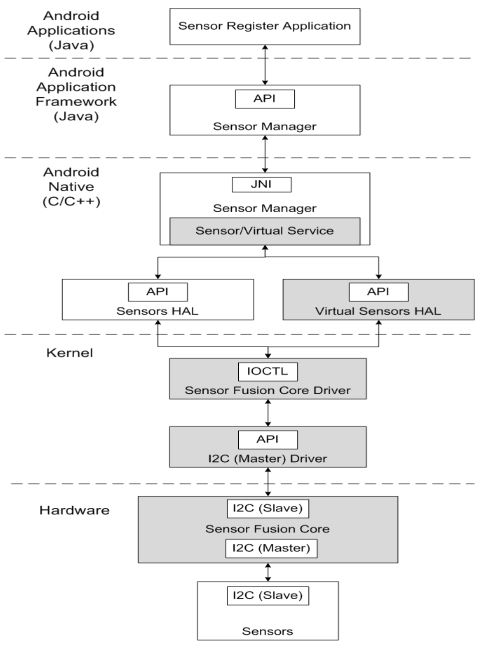


Fig. 5. The Position of Energy-efficient Mobile Platform in Android Stack

Our choice of external GPIO interrupt comes from the accelerometer sensor. Accelerometer sensor has its own clock and is always turned on monitoring accelerations. By exploiting the characteristics of acceleration, we use the accelerometer IC as the driving force to wake the sensor fusion core from OFF mode. Meanwhile, the ODR (Output Data Rate: the rate at which the sensor will interrupt with data ready) synchronization of other sensors with accelerometer are carefully designed so that we can perform the tasks the same as we configure a timer to wake up the lower power sensor fusion core.

Using this mechanism, we also benefit from the fact that the accelerometer IC already has the latest measurements that are ready to be retrieved loaded into the registers, whenever accelerometer sensor is triggered to function as a clock. The detailed synchronization procedure is shown in Fig. 6, and the key module is described in Algorithm 1.

Algorithm 1: ODR-SYN(ACCEL INTERRUPTS)

```
if Accelerometer interrupts with data ready then
    SensorManager retrieves the Accelerometer data;
end
if Other sensors registered for data then
    if Data requested as the same rate as Accelerometer then
        SensorManager retrieves sensor data;
    end
    else
        Delay data retrieval till the next Accelerometer interrupt;
    end
end
```

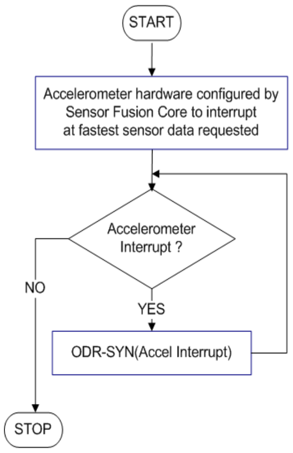

START
Accelerometer hardware configured by Sensor Fusion Core to interrupt at fastest sensor data requested
Accelerometer Interrupt ?
NO
YES
ODR-SYN(Accel Interrupt)
STOP

Fig. 6. Sensor ODR synchronization with Accelerometer

2) Power Voting Procedure: When designing the software architecture for the sensor fusion core, we need to ensure all software modules complete their tasks before the sensor fusion core could go into low power mode. Placing the system into low power mode before a task completes can cause data not to be reported, or corrupted. We design a module called the Power Manager that manages the power modes of software architectures. The Power Manager module uses a power-voting procedure shown in Fig. 7 and Algorithm 2 to determine when it is safe to place the sensor fusion core into *off* mode (LPM4).

Once the sensor fusion core is woken from the hardware interrupt (BUS interrupts, accelerometer interrupts), the power voting procedure starts. Our architecture system stays awake by allowing the first function call of all software modules to be voting for power state. Upon a software module completing its task, it will vote to enter low power. Our Power Manager monitors the votes from all the software modules to determine if it can place the system into low power mode. Take Algorithm Manager and Sensor Manager for example, for the movement algorithm, Algorithm Manager is voting to keep the system awake and then asks for sensor data from Sensor Manager. During this time, Sensor Manager will vote to keep the system awake for the retrieval of sensor data. As soon as sensor data is retrieved and delivered to Algorithm Manager, Sensor Manager will vote for going to low power mode. Once Algorithm Manager processes data, it will send data to Application Processor Interface and then will vote for low power mode. Algorithm Manager can also vote for low power mode in between requesting sensor and sensor data being delivered to Algorithm Manager to achieve the maximum reduction in energy consumption.

It is possible that one module can request low power mode and just in jiffies, the same module will vote for normal mode to perform that task. During the jitter period, if there is no other module requested for normal mode, then the system will go into low power mode just for jiffies. However, if there is another module requested for normal mode, then the vote from the first module will be overwritten.

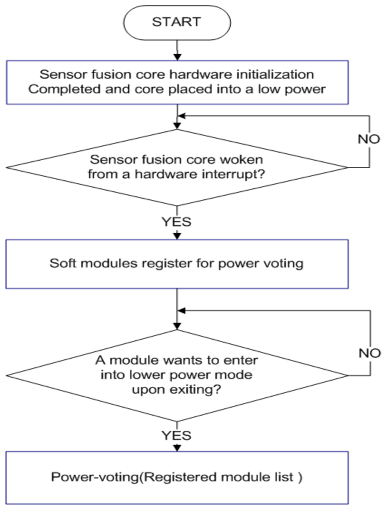

START
Sensor fusion core hardware initialization Completed and core placed into a low power
Sensor fusion core woken from a hardware interrupt?
NO
YES
Soft modules register for power voting
A module wants to enter into lower power mode upon exiting?
NO
YES
Power-voting(Registered module list)

Fig. 7. The Power Voting Procedure

Algorithm 2: POWER-VOTING(REGISTERED MODULE LIST Γ)

```
1  int Counter = 0; Counter records the number of votes for low power
   mode
2  foreach module $\tau \in \Gamma$ do
3      if $\tau$ votes to run at normal CPU speed then
4          PowerManager configures sensor fusion core to run at the
           system normal CPU speed
5      end
6      else
7          if $\tau$ votes to go into lower power mode then
8              PowerManager stores the vote for $\tau$ of low power mode;
9              Counter = Counter + 1;
10         end
11     end
12 end
13 if Counter == $|\Gamma|$ then
14     PowerManager configures sensor fusion core to go into low power
       mode;
15     Counter = 0;
16 end
17 else
18     PowerManager keeps sensor fusion core to run at the system
       normal CPU speed;
19 end
```

3) Virtual Sensor: Virtual sensors are sensors that can provide calculated context output instead of raw sensor data. They can remove heavy calculations done usually at application processor using raw sensor data and then just provide the calculated output to application processor.

Virtual sensor itself can run on the application processor. However, algorithms running on external processors, such as sensor fusion core, can also be treated as a new virtual sensor that Android applications can register to obtain results. By doing the same calculations at the sensor fusion core, same algorithmic output can be achieved at a much lower power. In our design, we created a virtual sensor for drive mode. The virtual sensor registers for drive mode algorithm running in the sensor fusion core and accepts the result back from the algorithms.

IV. Implementation Result

In this section, we discuss how our measurements of mobile platforms are designed and benchmarked based on a real implementation. For consistency, our measurement will be based on the drive mode detection example, although it is open to any sensor-related Android application.

A. Measurement Comparisons

For drive mode detection, we measure our data based on two comparative approaches discussed in Section III-A:

1) All location information was derived using a GPS sensor connected to the application processor.
2) Minor movement and non-movement algorithms were deployed to our energy efficient platform (sensor fusion core) to supplement GPS for location tracking in drive mode detection.

B. Measurement Methods

To be statistically accurate, we created a common method to measure energy consumption. An Android test application that contains the two comparative approaches for drive mode detection was created. Users can install the test application on their mobile devices and easily switch between the two approaches for testing under the same user-case scenario. We had approximately 250 users install the test application on different devices. We covered multiple physical distances ranging from 1 mile to 100 miles while tracking location. We also covered various scenarios like mild traffic to heavy traffic, multiple routes with few signal lights (like highways), and with many signal lights (like city). The test results will be shown in Table III. To perform the energy consumption measurements, we ran the same test application using our measurement platform and device discussed in the following section.

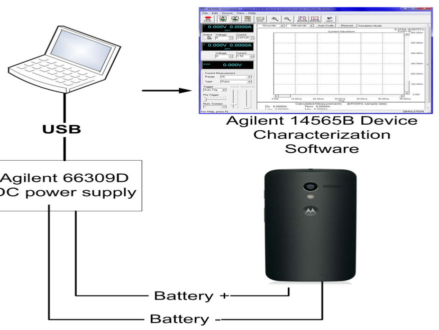
C. Measurement Platform and Device

Average energy consumption measurements were performed on the hardware that was modified by carefully removing the back housing and replacing the internal battery with soldered lines to battery "+" and "−" PCB connectors. As shown in Fig. 8, the battery lines of a modified mobile device were connected to an Agilent 66309D DC power supply. Once the system was wired up properly, we used an Agilent 1456B device characterization software on a personal computer to characterize energy consumptions for the experiments.

D. Measurement Results

On a general user case scenario, we measured the average energy consumption of drive mode detection using continuous GPS scanning vs. using the combination of GPS and sensor fusion core in our platform. The results are shown in Fig. 9 and Fig. 10, respectively. The vertical lines x_1 and x_2 in Fig. 9 and Fig. 10 capture the current drain for 20 milliseconds. In Fig. 9, the application processor wakes up every 1 millisecond for GPS tracking (seen as $\geq$ 100mA spikes in the current measurement plots), while Fig. 10 shows that the application processor wakes up every 5 millisecond for GPS fixes after the sensor fusion core detected the non-movement status of the mobile device. If the mobile device has not moved, the application processor will continue reading GPS fixes every 5 milliseconds till the sensor fusion core has detected the mobile device has moved. As soon as the mobile device moves, the sensor fusion core will notify the application processor to enter back into continuous GPS tracking mode. The average energy consumption in the diagram is the reported number between x_1 and x_2. It is clear that by significantly reducing the wake up frequencies of the application processor when the device in the non-movement status, our approach/design saves large amounts of current drain shown in Table I.

As the pattern of energy consumption in both of the approaches are repetitive, it is safe to say that the average total


USB
Agilent 14565B Device Characterization Software
Agilent 66309D DC power supply
Battery +
Battery -
Fig. 8. Measurement Platform

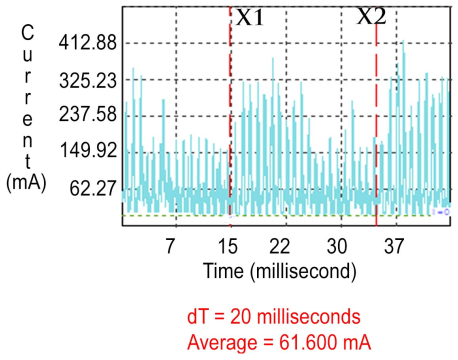


Fig. 9. Continuous GPS Tracking Measured at The Battery Terminal

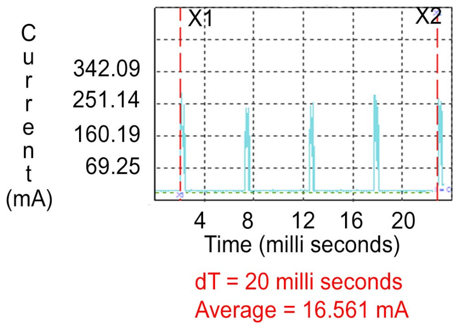


Fig. 10. Sensor Fusion Core Running Drive Mode Measured at The Battery Terminal

TABLE I
AVERAGE ENERGY CONSUMPTION MEASURED OF TWO DRIVE MODE DETECTION APPROACHES EVERY 20 MSECS IN NON-MOVEMENT STATUS

Drive Mode Approach	Average Energy Consumption Measured (mA)
Continuous GPS Tracking At Application Processor	61.600
Sensors At Energy Efficient Platform And GPS Tracking At Application Processor	16.561
Total Savings	45.039

saving of the energy consumption while in the non-movement status throughout the battery life would be 45.039 mA as well. Considering the battery in our test has 3.6 volts with 2200 mAh of capacity, the average battery life hours are 15 hours. By saving a total of 45.039 mA in the non-movement status, we can equally save 2.047% of battery capacity, which is prolonging 18.425 minutes of battery life hours shown in Table II.

TABLE II
ADDITIONAL BATTERY SAVINGS AND LIFE HOURS

Drive Mode Approach	Battery Capacity Used (%)	Battery usuage (mins)
Continuous GPS Tracking At Application Processor	2.800	25.200
Sensors At Energy Efficient Platform And GPS Tracking At Application Processor	0.753	6.775
Battery Capacity Savings (%)	2.047	
Battery Life Savings (minutes)		18.425

In reasserting and amplifying the empirical conclusions of our energy efficient platform, we conducted various user case scenarios under different driving conditions. The data in Table III reveals that our platform indeed reduces the energy consumption in the numerous user-case studies.

V. CONCLUSION

In this paper, we take the drive mode detection sensor-based application as an example and present an implementation of an energy efficient mobile platform. Fundamentally, our architecture supports any motion detection applications involving multiple sensors. The platform is portable and open to a large set of Android sensor-based applications and can easily port to IOS or Windows based devices. Proven in millions of consumer devices, our implementation is robust and data shows that the platform can save up to 73.11% of current drain and prolong battery life up to 18.43 minutes.

Our future work concentrates on the creation of a tightly coupled sensor subsystem that operates itself independently and remains loosely coupled to the main application processor. We will add low power GPS, WiFi and Bluetooth capabilities into the sensor fusion core. With minimal current drain, our subsystem will have complete awareness of its surroundings alerting the main processor only when substantial contextual events take place. Such a system has many applications from security, to alerting the user of points of interest, to the even more difficult problem of precise indoor navigation.

TABLE III
ENERGY CONSUMPTION FOR DIFFERENT USER CASES

User Case	Avg User Driving Time (mins)	Avg Energy Consumption Using GPS Continuously (mA)	Avg Energy Consumption Using GPS + Sensor Fusion Core (mA)
Location Tracking For 0-25 Miles Range Of Driving	27	27.62	20.25
Location Tracking For 26-50 Miles Range Of Driving	45	46.20	34.12
Location Tracking For 51-100 Miles Range Of Driving	79	81.11	63.33
0-50 Miles Range Of Highway Driving Without Traffic	47	48.25	38.10
0-50 Miles Range Of Highway Driving With Moderate To Heavy Traffic	69	70.84	36.78

REFERENCES

[1] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh, "A framework of energy efficient mobile sensing for automatic user state recognition," in *Proceedings of the 7th international conference on Mobile systems, applications, and services*. ACM, 2009, pp. 179–192.

[2] Z. Zhuang, K.-H. Kim, and J. P. Singh, "Improving energy efficiency of location sensing on smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 315–330.

[3] F. Ben Abdesslem, A. Phillips, and T. Henderson, "Less is more: energy-efficient mobile sensing with senseless," in *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*. ACM, 2009, pp. 61–62.

[4] C. G. Pendão, A. C. Moreira, and H. Rodrigues, "Energy consumption in personal mobile devices sensing applications," in *Wireless and Mobile Networking Conference (WMNC), 2014 7th IFIP*. IEEE, 2014, pp. 1–8.

[5] S. Li and H. Qi, "Pattern-based compressed phone sensing," in *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*. IEEE, Dec 2013, pp. 169–172.

[6] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A survey of mobile phone sensing," *Communications Magazine, IEEE*, vol. 48, no. 9, pp. 140–150, 2010.

[7] X. Sheng, J. Tang, and W. Zhang, "Energy-efficient collaborative sensing with mobile phones," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 1916–1924.

[8] M. A. Viredaz, L. S. Brakmo, and W. R. Hamburgen, "Energy management on handheld devices," *Queue*, vol. 1, no. 7, p. 44, 2003.

[9] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5. ACM, 2001, pp. 89–102.

[10] B. Priyantha, D. Lymberopoulos, and J. Liu, "Littlerock: Enabling energy-efficient continuous sensing on mobile phones," *Pervasive Computing, IEEE*, vol. 10, no. 2, pp. 12–15, 2011.

[11] D. Yee, R. Bickley, P. Zucarelli, and T. Keller, "Method and apparatus for control of an electronic system using intelligent movement detection," Mar. 14 2000, uS Patent 6037748 A.

[12] N. Patwari, R. O'Dea, V. Allen, M. Perkins, and M. Bourgeois, "Method and apparatus for location estimation," Sep. 5 2002, uS Patent 20020122003 A1.