18-847 Lab 3

Omar Serrano
omar.serrano@sv.cmu.edu
November 13, 2017

***Abstract*—We learn how to enable interrupts on the PowerDue because interrupts enable event-driven architectures which may be more power efficient than polling architectures, an essential charactersitic for wireless embedded systems.**

I. Introduction

Power efficiency is crucial for wireless embedded systems, because energy is a limiting resource and it can be expensive or impractical to replace batteries in wireless devices. Interrupts are a form of hardware-software mechanism that can enable an event-driven architecture, making a system not only more power efficient, but even more accurate than a polling architecture, because a polling architecture will be oblivious to events that occur whenever there is no poll.

This work explores the nuances of enabling interrupts on the PowerDue, by setting up the FXOS8700CQ sensor to interrupt the PowerDue when it detects a magnetometer event. Ultimately, an interrupt-driven application can result in a more complex architecture, because there is very tight coupling between the hardware and software; however, such an architecture will improve the power efficiency of the system because it allows the idle task to run for longer periods of time, or even to throttle down the clock rate, consuming less CPU cycles, and it allows the application to yield more accurate readings, because it does not miss events that would otherwise be missed in a polling architecture whenever such a system is in between polls.

II. Related Work

The iOS energy efficiency guide [1] provides a high level view of how event-driven architectures can improve the energy efficiency of the iOS phone. Despite the fact that the material is geared toward iOS app development, the material is relevant becuase it applies generally to wireless systems, and thus inspires our motivation for implemening an event-driven architecture. More specifically, [2] shows how an interrupt-driven architectures, a specific realization of an event-driven architecture, can lead to more power efficient and accurate WSNs, providing motivation for the interrupt-driven approach of our application.

To allow the PowerDue to be interrupted by the sensor, the Arduino reference for *attachInterrupt* [3] was relevant, because it shows the API of the function, and provides an example of how an interrupt handler can be registered with the PowerDue to handle interrupts on a given input PIN. Simply attaching an interrupt handler to the PowerDue is not enough to configure the system, because FXOS8700CQ also has to be configured to send an interrupt signal to the PowerDue, and [4] specifies how magnetometer registers have to be configured to enable FXOS8700CQ to send interrupt signals.

III. Approach

From a high level perspective, our approach to explore the nuances of enabling interrupts on the PowerDue is very simple, because it consists of three distinct aspects:

1) Enabling the PowerDue to be interrupted by creating and registering an interrupt handler with the PowerDue.
2) Calibrating the readings from the sensor to obtain threshold values that could be applied to the interrupts, to prevent being interrupted by the continuously changing magnetic field in any given environment.
3) Configuring the FXOS8700CQ to interrupt PowerDue whenever it detects magnetic events above the threshold values.

The three axes threshold values were computed by

$$x_{th} = |\overline{x}| + \sigma_x * m \tag{1}$$
$$y_{th} = |\overline{y}| + \sigma_y * m \tag{2}$$
$$z_{th} = |\overline{z}| + \sigma_z * m \tag{3}$$

where x_{th}, y_{th}, and z_{th} are the threshold values, $\overline{x}$, $\overline{y}$, and $\overline{z}$ are the averages of the readings, σ_x, σ_y, and σ_z are the standard deviations of the readings, and m represents a constant multiplier to prevent the sensor from interrupting on changes that can be expected from the continuous variation in the magnetic field.

IV. Experimental Results

A. Calibrating threshold values

Simply configuring the PowerDue and FXOS8700CQ with interrupts would not work because there is continuous variation in the magnetic field of a local environment, and hence the sensor would continuously detect magnetic events and send interrupts. To make the interrupts more useful, FXOS8700CQ can be configured to only interrupt when the mangetic readings from any of the axes cross a threshold value [4]. To compute x_{th}, y_{th}, and z_{th}, we added a multiple of the standard deviation of the readings to the absolute value of the mean of the readings. We experimented with different values of the standard deviation multiplier, m, and concluded that $m = 8$ resulted in an adequate sensitivity level for our use case, mainly being able to detect magnetic events caused by moving a cell phone, or magnetic strip, close to the sensor.

Essentially, calibration allows us to treat the average values of the readings due to magnetic variation as zero, as if we were

translating the offset value of the axes. FXOS8700CQ provides offset registers which can be applied to the readings after an event has been detected [4], and hence this is something that can be used to avoid having to manually translate values.

B. Enabling interrupts on PowerDue

To enable interrupts on the PowerDue, an interrupt handler has to be registered with *attachInterrupt* [3], and the interrupt pin has to be configured as an input pin. FXOS8700CQ can be configured to make the interrupt signal active high or active low, but we found the default setting to be active low, and hence configured the interrupt handler for this. Furthermore, we found it useful to detach the interrupt handler after receiving an interrupt signal, and to not reattach it until the magnetic event had been completely processed by the application, which in this case meant until *ProcessData*, a task for counting the number of events detected, had finished incrementing the counter for the number of events detected.

C. Interrupts on FXOS8700CQ

To enable the FXOS8700CQ sensor to interrupt the PowerDue when it detected mangetic events the threshold values, we had to configure the threshold register [4] with the threshold events and the interrupt pin. It was also necessary to add functionality to the FXOS8700CQ class representing the sensor, functionality that would allow us to check that an event had indeed occurred, and also verify that the values had crossed the threshold.

Without verifying that an event had occured, we would get spurios interrupts, especially during initialization of the program. More specifically, we would get one interrupt signal when the interrupt handler was initially registered, despite the fact that no event had been detected. Thus, to check whether an event had been detected, we would read the event detected bit in the M_THS_SRC register [4].

Looking at the event flag in M_THS_SRC did not completely prevent us from reporting values that were not above the threshold, because the sensor might update the magnetic values in between the event detection and when we reported the values. To prevent this from happening, we would only report the values if we confirmed that they were above the threshold.

For our use case, it was also useful to add a delay of 1 second after processing an event. Adding the delay allowed us to more intuitively associate an event with a reported value, and in our case an event meant moving a cell phone or magnetic strip close to the sensor. Without the delay, the application would report a burst of events, making it seem like there had been multiple events (indeed, it is logical to expect this, but we simply wanted to associate one reported value to an event, and this made it easier to do so).

V. Analysis

There are multiple aspects of this experiment that require further work, and in particular the following aspects could be improved to make the application more efficient and accurate.

1) Computing two threshold values for each axis

$$x_{th} = |\overline{x}| \pm \sigma_x * m \tag{4}$$

$$y_{th} = |\overline{y}| \pm \sigma_y * m \tag{5}$$

$$z_{th} = |\overline{z}| \pm \sigma_z * m \tag{6}$$

2) Reading all values from events detected
3) Have a functional system that does not use delays

Currently, for 1, the system is configured to only detect values that are greater than the threshold value, however, we also want to detect values v such that $v \leq |\overline{a}| - \sigma_a * m$, where v represents a magnetic reading, and a represents an axis (e.g., x). Clearly, these events lie outside the range of values that we expect from simple magnetic variation, and hence want to report them.

For 2, we want to make sure that we report all values, but as described in Section IV-C, we had to add functionality to verify the values before reporting them, because the sensor might update the values after an event is detected, but before the values from the event are reported. If FXOS8700CQ supports preventing the sensor from updating the values if there is no event, then it might just be a matter of configuring a register, otherwise the magnetic data might have to be read within the interrupt handler (but data would still be reported in *CollectData*). Regardless of what approach is taken, it is imperative that we report all values for events that are detected, especilly given the system is interrupted with notification that an event has occured - not reporting these values defeats the purpose of enabling interrupts and makes the application less reliable and accurate.

For 3, we want to remove any delays because they prevent the CPU from sleeping [1], [2], and because the delay may prevent the system from processing an event. We added the delay in order to associate one reading with one event (moving cell phone or magnetic strip close to sensor), but it is very likely possible to achieve the same result without having to resort to a delay. One alternative might be to use dynamic thresholding [2].

For example, after detecting a value above the threshold, the threshold value can be updated such that the sensor will only send an interrupt signal to the PowerDue if the next mangenetic event is greater the previous one by a given amount, and gradually taper-down to the original threshold value. This would increase the software complexity, but it would get us closer to the desired application behavior without having to resort to sleeping. In this particular case, there is a trade-off between sleeping and added complexity to adjust the threshold values dynamically, and hence it would be necessary to a thoroughu cost-benefit analsys to determine if dynamic thresholding is indeed more power efficient than simply adding a slight delay after processing an event.

VI. Conclusion

Setting up a system to be event-driven can be challenging because it requires very close cooperation between the hardware-software interface, there are subtleties around the behavior of interrupt handlers, and the hardware configuration and code can be more complex; however, such an architecture

can be essential for wireless embedded systems, in particular WSNs, which are expected to run a long time without getting battery replacements.

Ultimately, setting up the PowerDue and FXOS8700CQ to be event-driven results in a system that is more power efficient and accurate, because it allows the PowerDue CPU to remain in the idle state for longer periods of time, and to be more accurate, because we avoid the cost of missed events while the system is sleeping waiting to make its next poll.

REFERENCES

[1] [Online]. Available: https://developer.apple.com/library/content/documentation/Performance/Conceptual/EnergyGuide-iOS/index.html

[2] M. Malinowski, M. Moskwa, M. Feldmeier, M. Laibowitz, and J. Paradiso, "Cargonet: A low-cost micropower sensor node exploiting quasi-passive wakeup for adaptive asychronous monitoring of exceptional events," pp. 145–159, 01 2007.

[3] [Online]. Available: https://www.arduino.cc/en/Reference/AttachInterrupt

[4] "Fxos8700cq." [Online]. Available: https://www.nxp.com/docs/en/data-sheet/FXOS8700CQ.pdf