18-847 Wireless Software Architecture Lab3 Enabling Interrupts

Jun Ye
jun.ye@sv.cmu.edu
November 15, 2017

***Abstract*—This lab intends to help students get familiar with using interrupts for sensor data processing. Polling is very energy inefficient for processing data from sensors, and interrupt should be used for this purpose. This lab uses an interrupt handler to collect data when a signal from a NXP magnetometer sensor interrupt pin. The sensor's interrupt service routine is set as when several consecutive data above a set threshold are detected, the interrupt pin's state is changed. After this lab, I am more familiar with writing interrupt service routines and reading data sheet for enable and disabling certain interrupts.**

I. Introduction

Collecting useful data from a sensor requires using interrupts. When the sensor detects an interested signal, it can generate an interrupt and calls the interrupt service routine function. For this lab, we are interested in when the magnetometer sensor senses significant change in the magnetic field (e.g. a cellphone comes near the sensor). The magnetometer data is then collected and number of times the interrupt happens is also recorded.

II. Related Work

In order to figure out how interrupt is set up in the magnetometer sensor, I read the data sheet [1]. From the data sheet, I get to know that the sensor has a threshold function interrupt setup. I can set a certain threshold, above which the interrupt can be triggered. I can also set the latch mode, which allows me to set a number of times the data is above the threshold, then the interrupt can be generated.

In order to find out how interrupt on Arduino Due board, I read the Arduino reference interrupt section [2]. I get to know about attachInterrupt(), and detachInterrupt() functions. Use these functions enable any pin to be used as interrupt pins on Arduino. We can also set the interrupt as triggered when the pin signal changes, rises, or falls.

III. Approach

First, I added interrupt enable and disable and reading interrupt source register functions in the FXOS8700CQ.h and FXOS8700CQ.cpp files. This is done by writing to CTRL_REG4, CTRL_REG5, and M_THS_CFG registers and reading from M_THS_SRC register. Then it turns out that I don't only need to write to the M_THS_CFG register. Since we are not using the latch mode, we don't need to read interrupt source register to clear the interrupt.

Then I calibrated the magnetometer by reading the output when there is no interference. I included a calibration function which take 100 samples and calculate the average of these samples on x, y, z axises. Then I calculated the standard deviation for each axis and used the average value plus 5 times of the standard deviation as the threshold value. After setting up the interrupt on the magnetometer, I set up the interrupt on Arduino pin 51 using attachInterrupt() function.

Finally, I wrote a ISR function for sending a value to a global queue and a task to collect the magnetometer data when a value in the global queue arrives. It also counts the number of times the interrupt happened by incrementing a global variable.

IV. Results

For the calibration step, the x, y, z axis values' standard deviation is around 50. I set the threshold value as the absolute value of the average and add 5 times of the standard deviation. I am able to see the magnetometer data when interrupt happens and also see number of times the interrupts has happened.

V. Analysis

For calibration, I use 100 magnetometer samples, which takes about 30 seconds to complete the calibration process. I think 100 samples are enough to get an accurate average value and standard deviation. Because there is not much variance in the data. Increasing number of samples will not make it much more accurate, but it will increase the calibration time linearly.

Another improvement can be done for the ISR is considering the case of consecutive interrupts from the same source due to the small fluctuation of the magnetometer values. This happens when the interrupt source generate magnetometer values close to the threshold. This situation can be accounted by checking whether the interrupt value is higher than the threshold by a significant amount. If so, it can cause another interrupt. If not, the interrupt will not be triggered again.

Interrupt can be used to save a lot of power. From lab 2, we know that the Arduino due board's CPU consumes about 0.19 W power. If interrupt is used instead of polling for a 1 min interval, the amount of energy saved is 11.4 J.

VI. Conclusion

After this lab, I became more familiar with the magnetometer sensor's interrupt functions, and how to set interrupt service routine on Arduino. I also learned how to get useful information from reading data sheets. I realized that sensor reading are not very stable. When there is no apparent interference, the values still changes a lot. Because of these instabilities, careful setup and calibration has to be done beforehand in order to get accurate readings.

REFERENCES

[1] "NXP datasheet," https://www.nxp.com/docs/en/data-sheet/FXOS8700CQ.pdf, accessed: 2017-11-4.

[2] "Arduino Reference," https://www.arduino.cc/en/Reference/Interrupts, accessed: 2017-11-4.