18-877 WSSA Lab 4
Enabling Interrupts

Reese Grimsley Carnegie Mellon University November 21, 2019

***Abstract*—The purpose of this lab is to explore the usage interrupts and interrupt service routines. To maintain low average power usage, interrupts enable hardware to asynchronously branch the processor based on some electrical signal. In this lab, that signal is generated whenever the FXOS8700CQ magnetometer exceeds some threshold. The result of this lab is FreeRTOS code for PowerDué that counts the number of times a magnet is brought near the device.**

I. Introduction

This lab is designed to explore interrupts. Interrupts are one of the primary tools an embedded systems engineer may use to implement event-driven tasks and context switches. Built deep into the hardware of a microcontroller, interrupt handling will preempt the main order of execution such that events can be handled in real time. These events may be internal or external, *e.g.,* timer alarm goes off, memory buffer fills or a sensor indicates value has passed a threshold, Wi-Fi received a packet, respectively.

THe basic mechanic of hardware interrupts is that some piece of hardware monitors some digital signal for a characteristic like logic high, logic low, transition from high to low (rising) or low to high (falling). When this happens, the Nested Vector Interrupt Controller (NVIC) checks which interrupt was triggered, and performs a table lookup. This lookup returns a memory address to the interrupt service routine (ISR) or interrupt handler. The processor then goes to this memory address and executes whatever instructions are there. Generally, ISRs should be kept short because they will not preempt each other, meaning that a long interrupt may mean missed events.

Generally, ISRs make changes to flags or state of the program such that when the ISR completes, it resets the interrupt bit within the processor, and goes back to its main program to handle the remainder of the event. In this lab, I use FreeRTOS and setup the ISR to release a semaphore that another task is waiting for, namely the 'Collect Magnetometer Data' task. In FreeRTOS, there are generally special functions for OS operations performed within ISRs; I believe this is due to the way FreeRTOS manages memory in comparison to interrupt handlers.

While ISRs are a primary component of this lab, the sensor plays an important role as well — it generates the interrupt. The FXOS8700CQ can generate interrupts in several ways from either the magnetometer or accelerometer. I only use the magnitude of the magnetometer readings to generate interrupts in this lab. However, this would be difficult to do effectively without preprocessing or a priori knowledge of the magnetometer readings. Before setting up the interrupt, the device's readings must be normalized such that anamolous behavior is not confused with normal behavior and vice versa. This calibration step briefly looks at the statistical distribution of magnetometer readings to set interrupt threshold parameters.

In the experiments for this lab, I divided it into two parts: 1) magnetometer calibration and interrupt generation and 2) Arduino interrupt handling in FreeRTOS. The next section describes my approach and a few of the challenges I faced; concluding remarks are in the subsequent section. All code for this lab may be found at https://github.com/reese-grimsley/wssa-lab4.

II. Approach

This lab is divided into two. The first focuses on configuring the magnetometer to generate interrupt signals based on some calculated threshold. The second portion focuses on using that electrical signal to trigger an ISR, which will release a task to collect and process data from the magnetometer. Basic configuration and drivers for the magnetometer is reused from the previous lab

A. Magnetometer Configuration and Calibration

Calibration often entails collecting data from the sensor to make observations about the normal conditions of the sensor's environment. Since we are looking for times when a magnet is brought near the sensor, we primarily care about the relative changes in readings, not necessarily the absolute value in terms of Teslas. I first collect 20 sensor readings to determine statistical qualities, *i.e.,* mean and standard deviation. I then use these to calculate thresholds for interrupt event generation.

The first mode of interrupt generation I found was that whenever the signal goes above some X, Y, and/or Z axis value, it generates an interrupt. I then found that this only worked in one direction, *i.e.,* an interrupt is generated if the value goes above X_t+, but not below some analogous value X_t-, even though it is a large enough change from the mean to be interesting. The alternative is to use vector magnitude mode on the FXOS8700CQ . This allows a mean X, Y, and Z value to be entered, along with a threshold. The threshold defines the Euclidean distance from those means that the signal can go before an interrupt is generated. This seems much more appropriate for detecting magnet proximity. I set the mean or *offset* values as the mean of the signal (along each axis), and set the threshold to four times the standard deviation for each signal.

The only other things to configure were a set of bits pertaining to interrupt generation, interrupt pin, and vector magnitude mode enable. I chose not to let the IC update it's offset values internally, opting to keep the calibration data I set. It is worth noting that the magnetometer interrupt is latched

such that it will stay active low until a particular register is read. This could be used to control when new interrupts can be generated again, based on when sensor readings return to the normal range.

B. Arduino and FreeRTOS Interrupt Handling

Arduino generally makes interrupts painless by abstracting away many details about the nested vector interrupt controller and pin configurations. However, the interrupts handlers become less straightforward when used atop FreeRTOS, simply because it takes control away from the operating system and scheduler.

Once I was able to confirm that the interrupt signal was being generated and that my program would actually respond to it, I started toying with task control within the interrupt service routine (ISR). My first attempt was to suspend the data collection and data processing tasks and then resume them from the ISR. Even using the ISR version of the task resume commands, this appeared to break the program. The best way to handle this is semaphores. By making the ISR the only block of code that can give the semaphore (using the ISR specific function call), the interrupt drives the rest of the program that does data collection and processing for the magnetometer. In performing this portion of the experiment, I found that FreeRTOS and interrupts were sometimes finicky such that if the ISR was attached too early, the interrupt would run almost immediately and cause the program to hang. My code began to run consistently in the intended way after a series of small tweaks and code rearranging.

C. Question and Answer From the Manual

Are the [magnetometer values] constantly changing?

Yes, values routinely change, but typically in small amounts compared to the offset, *i.e.,* average.

What is the reason for the constant variations in the values?

The reasons for variation may be widely varied in prevalence and intensity. Flowing current incites magnetic fields, nearby ferromagnetic materials change field shape, sensor drift and quantization error, etc. are all possible reasons for this. While quantization error is typically thought to be on the order of a single bit, the real ADC in an MCU like the ATSAM3X8E (used in the PowerDué may routinely have 17 bits of error, based on an investigation of the datasheet). However, it likely that the ADC on this sensor is better tuned due its importance to the fidelity of the product.

How do you set the [interrupt threshold] level?

This depends on the mode used for interrupt: single axis or vector magnitude. For single axis, an interrupt can be generated based on the reading for that single axis compared to the value in the offset register, e.g., FXOS8700CQ_M_OFF_X_MSB and FXOS8700CQ_M_OFF_X_LSB for the X axis. The interrupt can be generated for X, Y, Z axis as an OR or AND operation between the three. Unfortunately, this does not use the actual variation effectively, such that if the variation is $\pm$ 5 and the interrupt is triggered if the value along X exceeds a threshold of 10, it would not similarly trigger if X went below -10. Clearly this has an issue with single-sided sensitivity; the fix is to use the vector magnitude mode. Simply put, this measures the distance between the sensor reading and the offset values placed in registers like FXOS8700CQ_M_VECM_INITX_MSB. Since the interrupt value, placed in FXOS8700CQ_M_VECM_THS_M/LSB, is based on Euclidean distance, it is able to trigger based on variation in an arbitrary direction.

How do you ensure that the variable signal itself will not trigger an interrupt?

The first step to solving this is to understand the variation of the signal. The standard deviation is the obvious measure to use. In my case, I decided that a threshold of five times the standard deviation is a robust threshold, and would not lead to many false negatives due to the sensitivity of the sensor whenever a metal or magnetic object is brought near it. *How do you adjust the default offset of the axis values?*

The offsets are adjusted by writing registers such as FXOS8700CQ_M_VECM_INITX_MSB and FXOS8700CQ_M_VECM_INITX_LSB.

How long would you collect data from the magnetometer?

Data should be collected until it's level returns to values below the interrupt trigger. Once readings return to normal, the register holding interrupt values must be read in order to clear the interrupt.

III. Conclusion

In this lab, I learned more sensor calibration and interrupt handling in Arduino/FreeRTOS. I set up a system that counts the number of times a magnet is brought near a sensor such that the processor can sleep when no magnet is present. This operation is fundamental to low power systems, in which power-hungry devices are kept off or in standby until needed.

References