Debugging Process for Dysfunctional Interrupts

**The Problem**

The Adafruit Feather M0 was having issues waking from sleep and interpreting interrupt signals in combination with the RocketScream lowPower.h library. Often the software interrupts would not trigger, would trigger twice for each interrupt signal, or, when used in combination with standby(), trigger once and then turn the device off indefinitely.

**The Process**

Debugging started with trying to understand the interrupts themselves. Interrupts are processes that are configured for when a peripheral device sends a certain signal to the microprocessor. The interrupt contains a call to an Interrupt Service Routine (ISR), a short function which will be executed if the expected signal is seen on the designated interrupt pin. On the Feather M0, interrupts can be attached to any pin except pin 4.

An initial issue that arose was that the interrupt signal was being modulated well after the signal should have returned to its normal state. Upon further inspection, this was due to a debugging process that used the LED builtin to tell when the device was awake. The LED builtin is connected to pin 13, and the interrupt was as well, so this caused an unexpected interaction when writing a signal to the LED and caused the interrupt to trigger indefinitely. It is recommended not to use pin 13 if you would like to write to the LED pin for debugging purposes.

The next problem encountered was that, when debugging without standby(), the ISR would be called twice for each time interrupt signal received by the M0. The accelerometer is configured to send a 10ms digital low pulse to the M0, and the interrupt was attached to trigger when low pulse is received on the interrupt pin. A variation of attaching the interrupt to trigger when receiving a RISING signal caused the ISR to only run once. This indicates that the attached interrupt was triggering twice on the logic low signal: the first time causing the ISR to be called as expected, the second time calling the ISR right when the interrupt was reattached. This was fixed by attaching the interrupt to an event that can only occur once for every interrupt signal.

When standby was introduced, the device would wake once on being interrupted the first time, then go into standby and not wake again. At this point in time, the interrupt was disabled within the ISR and was supposed to trigger on a logic low signal. This configuration followed the structure of interrupts used in Project LOOM. The solution to this problem was to detach the interrupts outside of the ISR in the code. Within the ISR, it is important not to have too many functions, and not to update variables larger than 8-bits. This is because having a longer ISR can let the interrupts trigger with the routine and cause an infinite loop, hanging the process and rendering the device useless. The explanation for this fix is not exactly clear, as project LOOM was having problems with the interrupt detach being outside the ISR, whereas SlideSentinel had the exact opposite problem. A probable explanation is that the ISR is called after attaching the interrupts and before putting the device to sleep, thus detaching the interrupts right before the device is put to sleep.

An ongoing problem is that the device will not print to the serial monitor after waking from standby. The workaround being implemented is detaching the USB device and closing all serial coms just before attaching the interrupt to wake from standby, and reattaching the usb device and serial coms just after the device wakes and the interrupts have been detached. After the usb device has been reconnected, the serial monitor must be closed and reopened. This issue only really affects debugging.

**Notes for Using Interrupts**

* Be sure that if using standby and detaching the interrupts outside the ISR, no functions or operations should be called between the calls to attaccInterrupts(), standby(), and detachInterrupts().
* Contain as few functions as possible within the ISR. Use Boolean status flags within the ISR and then apply functionality attached to those flags in loop() instead of changing large variable or running multiple functions.
* Make sure that the pins being used for interrupts are not being used for any other peripheral functionality.
* If interrupts are not working properly, try both detaching them inside of the ISR and outside too.
* When attaching interrupts to be used with standby(), be sure to use either logic LOW as a trigger or logic HIGH. RISING and FALLING interrupts require timers to operate, and will not work in conjunction with standby() as these timers are disabled.
* Serial monitor is finnicky when used with standby(), making debugging hard. If you can’t get the monitor to work, turn on the LED builtin just after the device wakes and interrupts are detached, then turn it off just before attaching interrupts and putting the device to sleep. This gives you are very reliable way of seeing when the device is active.
* A comment within the rocketscream library readme about patching the Arduino SAMD core is misleading, if your libraries are up to date for the device, the patch has already been implemented.
* Any variable that you want to change in the ISR must be volatile, otherwise its values will be restored when the ISR quits

**Basic Standby with Interrupt Steps**

1. Define a volatile bool global status flag
2. Configure a wakeup pin as INPUT\_PULLUP or INPUT\_PULLDOWN in begin().
3. Attach interrupts at the end of begin
4. Close all serial coms in loop
5. Detach the USB device in loop
6. Conditionally attach interrupts in loop based on the flag being true, set flag false within same if statement
7. Call standby
8. Set desired flags true in the ISR
9. Detach interrupts in loop, just after standby is called
10. Reattach usbdeivce
11. Restart Serial