Bugs and their fixes

**Code hangs when running xTaskNotifyFromISR from within ISR:**

The ISR must have a higher priority than the task it is notifying. If the task has a higher priority, the code will switch to the task immediately and will not complete the ISR. Fix is to change the ISR priority to be a low number.

**Code hangs at configASSERT( ( portAIRCR\_REG & portPRIORITY\_GROUP\_MASK ) <= ulMaxPRIGROUPValue );**

Ensure that all preemption priorities are zero. If they are non-zero, FreeRTOS does not work properly and will configAssert (error and loop forever). If all priorities are zero then NVIC\_SetPriorityGrouping( 0 ); can be used before the scheduler starts to ensure they are all zero.

**ASM330 not responding**

The ASM330 initially was not replying over SPI. After writing the NSS pin high during initialisation the sensor started working. Since the code used is from a generic library, a library agnostic solution is to set the default state of the NSS pin to HIGH in the .ioc configuration by setting the pin’s GPIO output level to HIGH.

**Cannot run STM32H743 past 192MHz. I get an error in HAL clock configuration:**

**f\_mkdir function hangs forever in SDIO 4 bit mode:**

This problem seemed to be fix by increasing the clk divide factor to 3. Any value lower and the code breaks.

**FreeRTOS gives up priority of data logging task after f\_stat:**

The cause of this is still unknown however, it appears to be specific to when using FreeRTOS. The SD card words correctly with a bare metal implementation

Update: Changing clk divide factor to 3 appeared to fix this issue.

**USB device recognised but mass storage not initialised:**

This was caused because the SDIO clk speed was higher than the USB clk speed. This was fixed to lowing the SDIO clk divider to 4 (for 192mhz sys clk speed).

**MS5611 did not respond to SPI commands:**

Either during the soldering (too hot) or otherwise, the sensor appears to have been damaged. A new sensor soldered at 300C worked correctly.

**ADC not reading correct value:**

The continuity ADC seemed to be reading out a high value when it is supposed to be low. It seems that the source impedance for the adc may be too high (from the voltage divider) resulting in the ADC not receiving sufficient current to charge up. The fix to the solution is to average the read values. I averaged 200 reads. Adding a 100 tick delay after starting the adc also helps to stabilise it. *The instabilities appear with input voltages closer to 3.0V. At 4.2 volts, the instabilities are minimal. This is because the in an open circuit, the ADC reads 45,000 at 4.2Vin while it reads closer to the 40,000 ADC value threshold (to determine good ematch) when at around 3.0V.*

**Magnetometer not reading values:**

This is likely explained by this forum post:  
<https://community.bosch-sensortec.com/t5/MEMS-sensors-forum/BMX055-Magnetometer-Data-Ready-Interrupt/td-p/13163/page/2>  
The magnetometer needs to be read in order to clear the interrupt. Its possible that some sort of timing issues with the interrupt firing during a read sequence may cause issues. To fix this, I’ve just slapped the BMX055\_readCompensatedMag(&bmx055, bmx055\_data.mag); function in the bmx055 gyro read block as well so that it’ll clear any mag interrupt.

**Unable to connect to flash memory over mass storage device:**

It appeared that the mass storage code was initialising before the SD card code which caused errors. To fix this, I added a check to make sure that the SD was initialised before the mass storage inits. The mass storage inits in usbd\_storage\_if.c.

**ASM330 interrupt stops firing after a period:**

It seems that after a second or so, the ASM330 data ready interrupt stops firing. When I add a manual read of the accelerometer and gyroscope in a separate function that runs in a loop with a 1 tick delay, it seems to allow the interrupt to start firing again. I suspect that it is because the ASM330 requires a read to clear the interrupt and if the interrupt fires, but the task is not notified, it will never fire again to re-notify the task.

This was indeed the case and I fixed it by setting the data ready interrupt to be pulsed instead of latched:  
asm330lhhx\_dataready\_pulsed\_t val;  
val = ASM330LHHX\_DRDY\_PULSED;  
asm330lhhx\_data\_ready\_mode\_set(&asm330->dev\_ctx, val);

**Packet streaming from sensor is intermittent and drops out:**

I added HAL\_Delay(1); in LoRa\_transmit when waiting for the TX IRQ flag to be set. I also added:  
taskENTER\_CRITICAL();

uint8\_t res = LoRa\_transmit(&LoRa\_Handle, send\_pkt, len + 15, 1000);

taskEXIT\_CRITICAL();

to prevent context switching to a higher priority task such as reading from sensors that use the same bus.

What I think was happening is that the code began a SPI transmission with the RFM95 but before it could complete, a sensor triggered an interrupt which flagged a task of a higher priority. This caused the scheduler to switch context to the sensor read task which resulted in a read on the SPI bus of a different sensor. The context would then switch back to the RFM95 read and try to finish the transaction. Since both the sensors and the RFM95 are on the same SPI bus, its likely this action caused a lot of issues.

**Semaphore always times out:**

This occurs because the STM32 code generation has a bug where it does not create a new semaphore with an initial number of tokens greater than zero.

The default generation looks like:

deploymentPinsSemaphoreHandle = **osSemaphoreNew**(1, 0, &deploymentPinsSemaphore\_attributes);

when in fact is should be:

deploymentPinsSemaphoreHandle = **osSemaphoreNew**(1, 1, &deploymentPinsSemaphore\_attributes);

**State streaming stops and starts randomly:**

This error is when the packet streaming seems to stop and start randomly and not follow the periodic timing that it is set to follow. After running a SWV profile on the STM32, I was able to see that the SDMMC\_GetCmdResp1 was being called ~70% of the time. It turns out that this function is called from the SD\_get\_free\_space\_kB() function call and was taking a very long time to compute. To fix this, I moved this function and the battery voltage calculation to the default task which runs every 10 seconds. Now instead, the state is updated asynchronously and it’s latest value is packetized by the steaming function and send over LoRa.

Although, the fix above dramatically increased the length of time that the streaming worked, it did not allow it to work indefinitely. By disabling all other devices on the same SPI bus as the RFM95, the performance increased dramatically. This suggests that noise/signal integrity issues from other devices are causing the issues. To remedy this, I decreased the bus frequency from 6MB/s to 1.5MB/s. **In future revisions of Strelka, putting the RFM95 on its own SPI bus would be preferrable.**

With the other devices on the SPI 2 bus disabled, the streaming was able to operate uninterrupted for 1 hour. This very strongly suggests that the other devices on the SPI bus are causing issues for the RFM95.

Disabling the BMX055 allowed to system to run for 20 minutes before stalling. Disabling the ASM330 allowed the system to run for about 50 mins. This cause of this may be because the ASM330’s I2C was not disabled using asm330lhhx\_i2c\_interface\_set(&asm330->dev\_ctx, ASM330LHHX\_I2C\_DISABLE); which may have meant that every time CS went low, the ASM330 thought it was in I2C mode and started doing some random stuff.