Lab 4

Part 1

For part 1, I first researched the cortex_m crate to see if there was any support for safe pushing/popping to the stack. Although I didn't find anything to help me safely do that, I did find a cortex_m method that safely reads that main stack pointer, which I used in my code instead of reading the value within my unsafe assembly blocks to try to minimize my use of unsafe code. Then I used the programming manual to determine what instructions I needed to use to set up/operate and their syntax. One issue I came across quite early in the process was that 3 sets of curly braces were needed to format the "reglist" operand for the PUSH and POP when using the "in/out" register feature in the inline asm. For reading and writing to MSP, I used the MRS and MSR instructions respectively, as the MSP is a special register and therefore requires special instructions for reading and writing.

After finding the right instructions to use for operating the stack, I then had to choose a location for my initial stack pointer.

This was probably the most difficult part of Part 1 for me. At first, I thought that the range of SRAM we could use was from 0x24000000 to 0x3FFFFFFF as I didn't realize at first that the memory map on page 28 of the programming manual dictated what type of memory every possible address that could be created with 32 bits would point to. Once I realized this, and that on the board we use actually only had 128 kB of SRAM, it became clear that the actual range of addresses was 0x20000000 (the start of the SRAM addresses) to 0x20020000, which is 128 kB worth of addresses. Then, I picked 0x20000000 as my stack address. However by the time the interrupt handler was called, the led_state variable on the stack had been erased and replaced with 0xfff…. Assuming the board was using those memory addresses for other purposes, I switched to using 0x20000100 as my initial stack pointer. This finally ended up working.

The overall structure of the code was very simple. I first initialized the TIM3 interrupts and LEDs in main. I then also set up my stack in main by switching MSP to my initial stack pointer (0x20000100) , pushing the current led state (off), and switching the stack pointer back to its old value. Each time the TIM3 interrupt occurred (which was every second), I read the LED state with a similar procedure as above, but popped the led_state off the stack instead of pushing it. According to that state, I turned the red led on/off, updated the led_state variable, and stored it back on the stack.

In order to reduce my use of unsafe rust features, I put all variables that needed to be globally accessible in mutexes. This ensured that no race conditions would occur.

Source Code for Part 1

I used my Lab 3 Part 2 main.rs code to initialize the red LED, TIM3 interrupt, and the necessary clocks. I also used this code to help me put different variables into mutexes and access variables within the mutexes. I added the body of the TIM3 interrupt and the stack set up code in main to the previously mentioned parts of my Lab 3 Part 2 code to create my Lab 4 Part 1 main.rs.

I also used my makefile, Cargo.toml, memory.x, and .config file from my Lab 3 Part 2. I used all these files unchanged except the makefile and the .config file. I changed the target of my makefile to be "part1_cargo" and changed make debug to not load from a specific file. I also added a "runner" setting to my .config file so I could use "cargo run" to launch gdb. I additionally updated their version of the cortex_m crate I was using, so I added the new dependencies of that to my .config file.

Source code for my Lab 3 Part 2 code can be found in my Lab 3 report.