3 – IOCTL driver

Objectives
- Development and test of an ioctl driver.

Description
You must complete this example driver with an ioctl method that allows passing messages between the driver and a userspace application. All developments are made natively on Exynos boards running an Ubuntu distribution.

Documentation:
https://www.hardkernel.com/shop/odroid-xu4-special-price/
http://magazine.odroid.com/odroid-xu4/

1.1 Test of sample project
A template ioctl project is already available on the Exynos board in directory Development/LABS/TP_IOCTL. Test this project, describe the content and explain the test script.

1.2 Development of an ioctl method
Complete the ioctl method to be able to send a message to the driver (using request code IOCTL_SET_MSG) or to read a message from the driver (using request code IOCTL_GET_MSG).

Test previous ioctl method with an application that will send a message to the driver, and then read back the message for verification.

Explain your code, tests and results.

1.3 Application: ioctl based performance monitoring
Exynos boards include ARM Cortex based processors that have specific registers to monitor execution parameters such as the number of processor clock cycles, cache misses, branch mispredictions, etc. The sample ARM assembly code provided in the appendix shows an example configuring PMNC (Performance MoNitor Control) registers to measure the number of clock cycles of a printk statement.

A technical documentation of PMNC registers is available online:

The goal is to develop an ioctl method that will be called perfmon_ioctl to measure the number of clock cycles between two triggering events corresponding to the following request codes:
- IOCTL_PERFMON_START: to reset and start the performance counter.
- IOCTL_PERFMON_STOP: to stop the performance counter and report the number of clock cycles measured between IOCTL_PERFMON_START and IOCTL_PERFMON_STOP.

Write also a userspace program to test this driver.
Explain your code, tests and results.
void testcounters() {
    u32 val1, val2;

    // Disable all individual counters
    asm volatile("mov r0, #0x8000000F");
    asm volatile("mcr p15, 0, r0, c9, c12, 2");

    // Disable the PMNC
    asm volatile("MRC p15, 0, r0, c9, c12, 0"); // Read PMNC
    asm volatile("ORR r0, r0, #0x0"); // Disable
    asm volatile("MCR p15, 0, r0, c9, c12, 0"); // Write PMNC

    // Select register and event
    asm volatile("mov r0, #0x0");
    asm volatile("mcr p15, 0, r0, c9, c12, 5");
    asm volatile("mov r0, #0x55");
    asm volatile("mcr p15, 0, r0, c9, c13, 1");

    // Enable all individual counters
    asm volatile("mov r0, #0x8000000F");
    asm volatile("mcr p15, 0, r0, c9, c12, 1");

    // Enable PMNC
    asm volatile("MRC p15, 0, r0, c9, c12, 0"); // Read PMNC
    asm volatile("ORR r0, r0, #0x7"); // Enable and re-set
    asm volatile("MCR p15, 0, r0, c9, c12, 0"); // Write PMNC

    // Read CCNT (Cycle CouNT) Register
    asm volatile("mrc p15, 0, %0, c9, c13, 0" : "=r" (val1));
    printk(" CCNT = 0x%08x\n", val1);

    // Run test function
    printk("<1>Profiling printk!\n");

    // Read CCNT (Cycle CouNT) Register
    asm volatile("mrc p15, 0, %0, c9, c13, 0" : "=r" (val2));
    printk(" CCNT = 0x%08x\n", val2);

    printk(" EXEC TIME = %d CYCLES\n", val2-val1);
}