## Assignment 1 – Linux Kernel Driver and MSP-430 NVM (Due by 2024/11/21)

1. Writing a Simple Linux Driver and Recompiling the Kernel (50%)

- Setting up the Environment: Verify if kernel headers and development tools are installed on the system. (Command: `uname -r` and `sudo apt install build-essential linux-headers-$(uname -r)`). Provide your screenshot. (5 pts)

- Prepare a simple "Hello World" kernel module. You are asked to write a simple "Hello World" kernel module and a **Makefile** to compile your kernel module. (10 pts)

Example code:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Student Name");
MODULE_DESCRIPTION("A simple Hello World module");

static int __init hello_init(void) {
   printk(KERN_INFO "Hello, world!\n");
   return 0;
}

static void __exit hello_exit(void) {
   printk(KERN_INFO "Goodbye, world!\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

- Loading and unloading the module. After compiling your module, you should use commands (i.e., `insmod` and `rmmod`) to load and unload your module. Moreover, you need to verify the module status by checking the kernel log (command: `dmesg`). (10 pts)
- Recompiling the kernel to include the module. Students are asked to clone the Linux kernel source and reconfigure the kernel to build the module as a built-in part of the kernel (not a loadable module). Verify the kernel includes the module by checking `dmesg`. (15 pts)
- Demonstration: Students must demonstrate that the module loads automatically at boot or via manual insertion (`modprobe`). Moreover, you should submit screenshots of their compiled module, kernel messages from `dmesg`, and a short report explaining their process. (10 pts)

2. Collecting and Processing Temperature Data on MSP430 FR4133 using FRAM (50%)

- Setting up the temperature sensor. Students are tasked with configuring the MSP430's ADC12 to collect data from the internal temperature sensor. Implement the function readTemperature() to periodically collect temperature data. (10 pts)

Code skeleton:

```
#include <msp430.h>

void initTemperatureSensor() {
   ADC12CTL0 = ADC12SHT0_2 | ADC12ON;
   ADC12CTL1 = ADC12SHP;
   ADC12MCTL0 = ADC12INCH_10;
   ADC12CTL0 |= ADC12ENC;
}

unsigned int readTemperature() {
   ADC12CTL0 |= ADC12SC;
   while (!(ADC12IFG & BIT0));
   return ADC12MEM0;
}
```

- Set up FRAM storage for the temperature data. Ensure that the temperature data collected is written into FRAM after every reading. Students should ensure that each time the system reads a new temperature value, it is stored into FRAM immediately. (10 pts)

- Modify the system so that after a power loss, the last temperature data stored in FRAM is correctly retrieved and used for further processing. This ensures that no data is lost. Implement a power recovery mechanism where, upon reboot, the system checks FRAM for the last valid temperature value and uses it for processing. (10 pts)

- Use the temperature data retrieved from FRAM to implement a simple application, such as comparing the current temperature to a set threshold. If the temperature exceeds a specific threshold, trigger an LED on the board to indicate overheating. (10 pts)

- Demonstration: Students should test the system by simulating power loss and recovery: Collect temperature data, power off the system, and power the system back on and verify that the stored temperature data is correctly retrieved and used. (10 pts)