Introduction to Mechatronics (ME/AE 6705)

Lab Assignment 6

Data Acquisition on the MSPM0

Objective

This purpose of this lab is to learn how to use onboard flash memory to save data acquired through analog to digital conversion or other means. A secondary objective is to learn how to print binary data in ASCII format using the `sprintf()` function.

Deliverables and Grading

To get credit for this lab assignment you must:

1. Submit a typed report as a PDF file to Canvas, answering the questions at the end of the lab (2 pages max, can be shorter). This is due at the beginning of class on the due date specified on Canvas. (20 points)

2. Demonstrate proper operation of your code to TAs or instructor during office hours or demo hours. You must write your own code for this lab – no lab groups are allowed. (40 points)

3. Submit the commented final version of your code on Canvas (single .c file containing your main() function, do not submit header files, etc. You should only submit a single file.) (Pass/Fail)

Setup

This lab requires Code Composer Studio, the MSPM0, and the temperature sensor circuit built in Lab 1. The lab uses onboard features of the device such as flash memory, ADC, and UART. The MSPM0 has 64KB of flash main memory. This lab makes use of main memory to store data acquired by the ADC.

Note: To create a new CCS project for the MSPM0 target device, follow the instructions in the document entitled "How to create a new project for the MSPM0.pdf" on Canvas under MSPM0 Documents.

Problem Statement

The goal of this lab is to use the microcontroller as a data acquisition device. The program you write will have two modes: data acquisition mode, and data output mode. In data acquisition mode, the program will collect a total of 30 data samples from the ADC at 1 sec intervals and store it in flash memory. In data output mode, the program will print out the stored data from flash memory into the putty terminal window.

Background

Flash Memory:

The memory space on the MSPM0 is a 2GB address space divided into several zones. The first 128kB zone is called the Code zone and consists of Flash memory. The Code zone contains both ECC (error correcting code) and non-ECC regions. ECC uses extra hardware to ensure that there are no errors when writing data to memory, but is also slower than non-ECC memory. The Code zone starts at address 0x0000_0000, i.e. at the beginning of the entire memory region, and is used to hold both your code and user data that you choose to store there.

The Flash memory space on the MSPM0 is comprised of a single bank. The bank of flash memory is divided into 66 sectors. Main memory takes up 64 of these sectors and is the space we will be using in this lab.

Each sector in main flash memory can be individually programmed and erased. Driverlib API's are available for programming and erasing flash memory sectors, as discussed in the lecture on flash memory. The output of these API's is true if the operation is successful and false if the operation fails. If the operation fails, it is a good practice to trap the microcontroller in an infinite loop which does nothing. It is rare that failure occurs but nevertheless this is good practice.

Sectors must first be unprotected in order to be erased and programmed. Sectors are automatically protected again once they are programmed. See the class lecture on flash memory for more details.

Hardware:

The circuit assembled/built in Lab 1 should be interfaced with the MSPM0 similar to the setup in Lab 5. Adjust the potentiometer used to tune the gain of the op-amp to set the gain such that the output voltage is always less than 3.3V for the full temperature range. Connect the output of the temperature sensor signal conditioning circuit to the pin corresponding to the ADC module used. Furthermore, **connect a ground wire from the signal conditioning circuit to the MSPM0.**

Software:

Your program should proceed in the following steps:

1. First, you should wait in a loop until the user presses one of the buttons. Pressing button S1 should place the system in data acquisition mode, while pressing S2 should place the system in data output mode.

2. When the button is pressed, the mode is selected. If data acquisition mode is selected, the MCU should collect ADC measurements from the temperature sensor at a rate of 1 Hz (as in Lab 5). It should store the data in a standard array of floating point values. After it has collected a total of 30 measurements, it should

load (program) the collected data into flash memory. Program flash memory at the memory address `0x00001000`. After programming is complete, the program should then enter an infinite loop which does nothing, and an LED can light up.

3. If the user chooses data output mode, the data from memory location `0x00001000` (which points to the flash memory location where you wrote your data) should be printed to the putty terminal via the UART interface from Lab 4. Use the `sprintf()` function to convert the stored binary numbers into Ascii values. When you print the data, print the temperature as a floating point number with one decimal place (this can be achieved using the format descriptor "%.1f"). After you print the data, enter an infinite while loop that does nothing.

Note: If you reprogram the board between acquiring the data and printing it out, all data in flash memory will be lost (since the programmer erases it). Instead, use the RST button on the board (located at the top of the board, near the USB plug) to reboot it when you want to switch modes.

Tips:

- Use your code from Lab 5 as a starting point. Collect each ADC reading in the interrupt service routine and place it in a buffer (array). Once you have read all 30 samples, write this array to flash memory.
- After you reach 30 samples and are ready to write to flash memory, disable the timer interrupt to avoid the ADC continuing to sample.
- Store your data in flash memory as an array of floating point numbers.
- The memory address where you will be storing data should be defined in a macro as follows (this is ECC flash):

```
#define MAIN_BASE_ADDRESS (0x00001000)
```

- The memory address where you will be reading data from should be defined in a macro as follows (this is non-ECC flash):

```
#define MAIN_BASE_ADDRESS (0x00001000 + 0x00400000)
```

- The only interrupt you should use in this program is the TIMG rollover interrupt, as in the previous ADC lab.

Requirements

1. Successfully demonstrate the outcome of the program and all the required functionalities to TAs or instructor.

2. Submit the commented final version of the code on Canvas.

3. Answer the below questions and submit the typed report on Canvas.

Questions

1. Compare the use of the internal system oscillator (which is a digitally-controlled oscillator) vs using a Crystal oscillator for sourcing a clock. For what applications would you want to use a crystal rather than a digitally-controlled oscillator, and vice versa?

2. Explain what is the difference between MCLK, MFCLK and LFCLK.

3. What are the advantages of the general purpose timer TIMG compared to SysTick?