ECSE426 Microprocessor Systems

Winter 2013

Lab 2: Reading Temperature, Filtering, and Digital IO

Background:

A common operation in embedded microprocessor systems is sensor data acquisition and signal conditioning. If the signal is known to be noisy it is beneficial to oversample and then apply a low-pass filter to remove some of that noise. In Lab 1, you prepared a moving-average filter that will fit this purpose nicely.

In this experiment you will construct such a system, performing data acquisition and conditioning while providing a simple graphical output using the STM32F4-Discovery's LEDs.

To begin this experiment, you will first have to go to the desk on Trottier's 4th floor to claim your team's development kit. Development will be much like on the simulator but with a bit of hardware initialization code and the ST-LINKV2 SWD debugger.

Functional Requirements:

First, as a simple exercise in C, you will rewrite your Lab 1 *moving_average* function in C. For this you can declare a structure (C struct) to better manage your internal filter state. Obviously, this function will also need a new name. Both the structure and the function parameters are similar to the ones used in your assembly code. Use the same test vector as given before, however, this time it should be stored in a C array. You will compare the performance of the compiled C code with your hand-tuned assembly code and verify that they both perform the same computation. Tutorial I presented many techniques on performance evaluation; many tools in Keil's IDE view menu prove to be beneficial, performance analyzer, code profiling, tracing and the stop watch to name a few. Your C filter code can be verified in simulation mode as before.

Secondly, the STM32F4 microcontroller has a crude temperature sensor built in. The sensor measures the operating temperature of the processor. It is known to be very noisy and is connected to the chip's analog-to-digital converter (ADC). You are expected to periodically sample this ADC channel at 20Hz. You are **NOT** required to use DMA mode yet with ADCs. You will pass this sampled signal through your moving-average filter to get a less noisy signal. Using macros, you should be able to easily switch between the two moving average implementations at compile time to filter the signal; that is the assembly and C implementations. You need to experiment with the filter's window size by doing some analysis (extracting data out of Keil and plotting the filtered signals against the noisy one, the printf code to be given will be beneficial in this case). You might need to translate the readings to Celsius / voltages. You need to follow ARMs procedure for that as presented in the data sheet. Then, you are required to use the filtered 20 Hz signal to update the LED display, which will by default indicate the current temperature.

Each LED (there are four, each with a unique colour) will represent a 2-degree range, looping around as the temperature overflows the 8 degree range available using four LEDs in such a configuration.

Pressing the user button should switch the display to flashing mode where all four LEDs flash continuously. Pressing the button again reverts the code to the initial functionality of temperature sensing.

Testing

Naturally, most temperature readings should be below 30 C° but these sensors have lots of error and values above 30 are reasonable to see. You should extrapolate accordingly to whatever temperature range you have and decide on the ranges. To force the processor to heat up, you might wish to rub your thumb against it to warm it up. Better still, you might try another heat source: exhaust from laptops or the machines in the lab should be hot enough to change the temperature sufficiently -- though give it some time since the heat will be transferred more slowly through air. Just be careful not to short any pins on the board while handling it near metal cases.

Debugging

For the hardware part, you will no longer be running your program in simulation mode. We will be using real time debugging through the ST-LINK debugger (conveniently built-in on the discovery F4 board). To switch from simulation mode to hardware debugging, right-click on Target 1 in the left most project pane, choose Debug →Use ST-LINK debugger from the drop down menu. Click settings next to it and choose SWD protocol. Refer to Figure 1 below. Similar settings are used for programming: in the same window, click on Utilities, also choose ST-LINK Debugger from the drop down menu. Finally click settings→ Add → STM32F4xx Flash. Save and rebuild; now your project should be working. Figure 2 illustrates setting up the programmer.

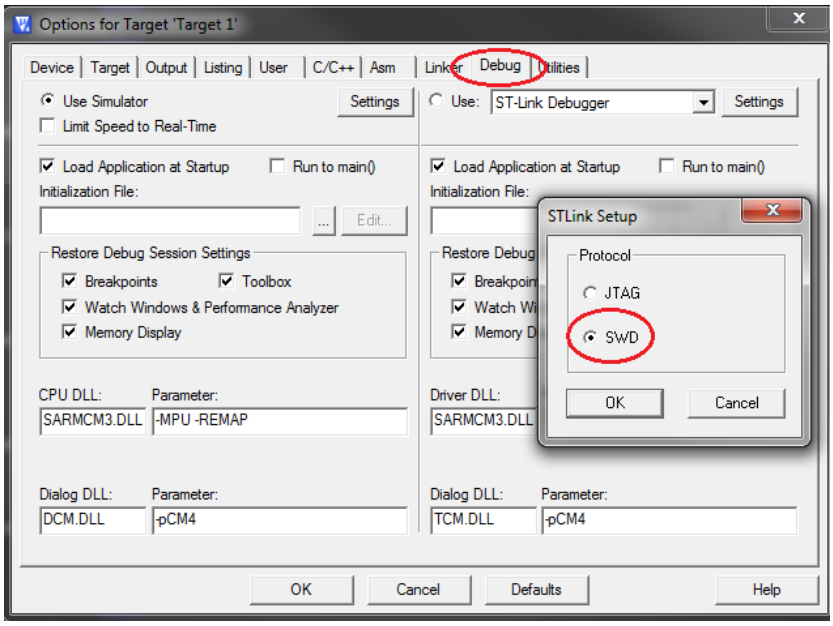

Figure 1 - Swetting up the Debugger

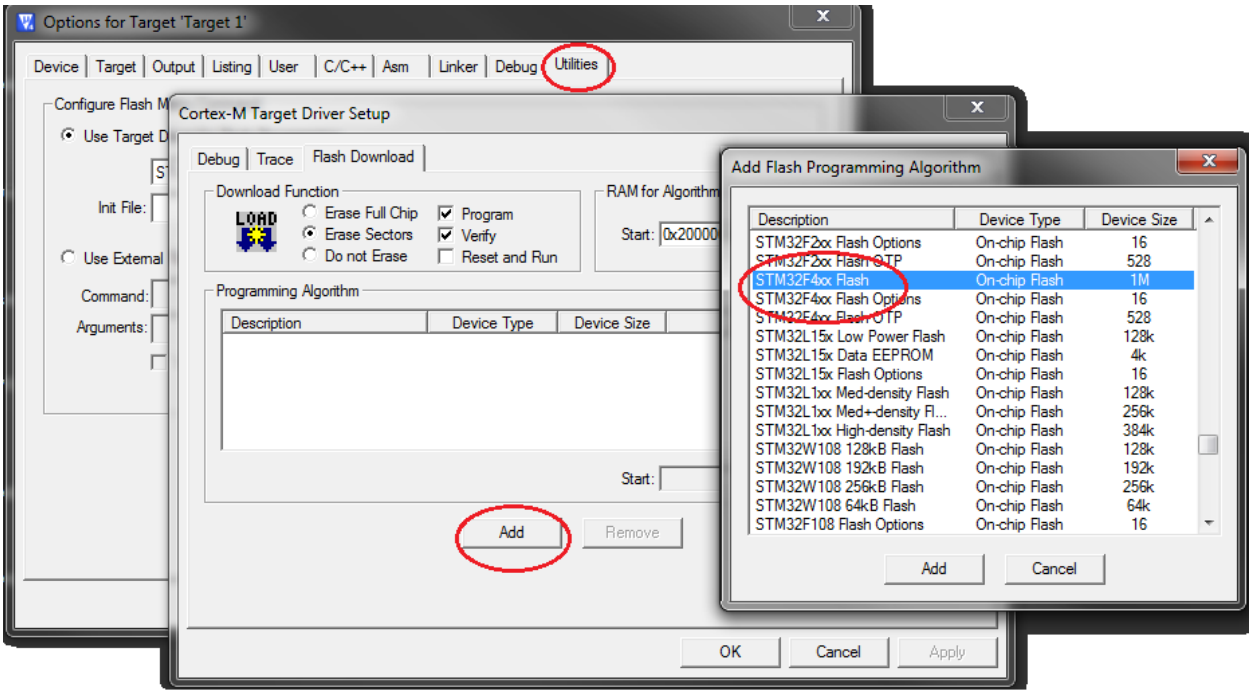

Figure 2 - Setting up the programmer

Hints

LEDs and Push buttons

The use of general-purpose IOs (GPIOs), ADC and the temperature sensors is fully described in the STM32F4 Reference Manual. To see how the user button and the LEDs are connected to the processor, please consult the F4-Discovery User Manual.

Timing:

Your timing must be exact. Approximations outside the tolerance of the 8MHz crystal are not okay. ARM provides a very nice peripheral for this: the SysTick timer. With a simple single-line initialization, it will execute an interrupt handler function with a precise period. Use this to create a timebase for your sampling.

Interrupts:

DO NOT perform the sampling in your interrupt handler routine. Interrupts happen in a special processor mode which should be limited in time. You can start the sampling process in the handler but do not wait for the conversion to complete. This can be achieved by setting a flag inside the ISR which can be observed from your regular application code once you return from your interrupt routine.

Don't worry about using other interrupts beside the SysTick timer. You can just sample (polling) the button and debounce it to detect button presses.

You will have a simple function "void SysTick_Handler(void){}" which will execute each period of the SysTick timer.

Examples Codes to get you started:

Many examples exist for various elements of this exercise. Browse the STM32F4 peripheral library examples for ADC usage, SysTick, and GPIO interactions. With an understanding of how to use the ADC, you can read in the STM32F4 Family Reference Manual about how to use the temperature sensor with the ADC.

Reading and Reference Material

- C Tutorial by Ben Nahill
- F4 Discovery User Manual
- STM32F4 Reference Manual. Chapter 10: ADCs and Temperature Sensor.
- Example codes. Available at http://www.st.com/internet/com/SOFTWARE_RESOURCES/SW_COMPONENT/FIRMWARE/stm32f4discovery_fw.zip
- Of course you are free to refer to any other posted documents which you see useful.

Demonstration

Demos will take place Friday, February the 8th. The exact time will be decided later. We expect that your code be ready by 1:00 P.M that day and that you will be ready to demo. No extensions allowed without penalties. You will lose 15% per day. Weekends are considered two days. That is, if you delay till Monday, you will lose 45% of demo grade (Saturday, Sunday and Monday).

Final Report Submission

Reports are due by Monday February 11that 11:59 P.M. The system will not accept any late submissions and it will close after that time. Submit the PDF report and the compressed project file separately. **Don't include your report in the archive.**

For this exercise, you should ensure that the following parts in your report are clearly visible. Be sure to follow the report writing guidelines and the report grading checklist. **This list is by no means the only requirements:**

1. High-level design description,
2. The details of the ADC initialization, and the way that the temperature sensor is read out.
3. The description on how you arrived at the mapping from the raw temperature sensor data to the temperature in degrees/voltages.
4. Explain how did you ensure that the readings are moderately accurate, and if you performed any calibration, how did you do it.
5. Description of the code using the LEDs and flashing them.
6. Use of buttons to initiate measurements and switch the output display.
7. Include all the source code in the online submission, as well as the Keil project data in a compressed project folder. The TA should open your project file and run the code instantly. Please ensure that your code is well documented.

Who will grade my report?

Due to the imbalanced load assignment to the TAs, it is not necessarily that the TA who demoed you will be the one grading your report. As such, do not expect to explain things in your demo and leave them vague in your report. It should be clear to any reader.