ECSE426 Microprocessor Systems

Winter 2017

Lab 2: Analog Data Acquisition, Digitizing, Filtering, and Digital I/O

Introduction

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 apply a low-pass filter to remove some of that noise. In this experiment you will construct such a system based on embedded temperature sensors. You will perform a manual data acquisition and conditioning while providing a simple graphical output using 7 segment LEDs. You will also construct an automatic data acquisition using ADC with DMA access.

Preparation

To begin this experiment, you will first have to go to the ECE labs counter on Trottier's 4th floor to claim your team's development kit (only people who have signed in groups can receive their kit). You will be receiving two boxes: the grey box has the development board while the blue box has all necessary components for all labs. The kits will be ready for you to pick up on ***Friday, February 3rd***. Try to get your kit as early as possible to start working on the experiment. You need to go in groups, present your IDs and sign forms to acquire your lab kit. You will be given a checklist for the items in the blue box. Make sure all components are there before signing the forms. You will be held responsible for any missing/damaged items at the end of the semester.

You will write your codes the same way you did with Lab 1 except for few changes in project setup. You need to switch from simulation mode to actual hardware mode by choosing the option of ST-LINKV2 SWD debugger. This should establish connectivity for kit programming and code debugging. Make sure you have **installed the ST-link** drivers beforehand (Check Tutorial 1 slides for link). Sometimes, if the kit does not get recognized, try changing the USB port or unplug/re-plug the board. If there are still problems, ask the TAs.

Tutorial II will cover in details how to use peripheral drivers based on the STM32F4 Cube HAL firmware. You will also learn about, debugging, reading the board schematics and basics of interfacing (not covered in slides). To learn more on embedded C programming, some slides as well as an embedded C tutorial will be posted for your reference.

The Experiment - Functional Requirements

In this lab you are required to do the following:

1. *Setting up temperature sensor and ADC:* The STM32F4 microcontroller has a crude built-in temperature sensor. The sensor measures the operating temperature of the **processor**. Modern processors often have built-in temperature sensors which are used for thermal monitoring. Should a

processor core heat up above certain threshold, either hardware or software measures are used to throttle processor speed to cool processor down. Though we won't be using the temperature sensor in such advanced capacity in this experiment, you will learn how to read this sensor and make use of the reading. You are required to write the code to set up the Analogue to Digital Converter (ADC) to digitize the temperature values by doing single conversion at a time. The sensor data should be acquired at a frequency of 100Hz. You are referred to the ADC section in the STM32F4xx datasheet, consult the tutorial and the STM32F4 Cube Documentation to get started with the setup.

2. *Setting up the sampling frequency:* To generate the required sampling frequency, you are required to use ARM's SysTick timer. **No software delays are allowed for the purpose of sampling**. Do **not** use any of TIMx peripherals yet. Only SysTick is allowed. Using SysTick timer entails two steps, setting up a required frequency and an associated interrupt handler code (see stm32f4xx_it.c). The SysTick interrupt body should be lightweight as described in the tutorial. You are encouraged to investigate the driver code of the function setting up the frequency. Note that this frequency is different than the sampling frequency of ADC part.

3. *Signal filtering:* Sensor data as any other signal is prone to noise, there are many sources of noise: electromagnetic interference, thermal noise and quantization noise due to ADC conversion to name a few. As such, one needs to filter these noise sources and minimize their effects. There are quite a number of filters used in all sorts of engineering problems. To keep things simple, you can use a simple FIR filter, the principle of which was covered in lab 1, to do the filtering. You are free to use your lab 1 implementation of the filter if you want, but you will need to modify it slightly for real time filtering. Make sure you have a correct functioning filter which can handle real time data, this means you will be reading inputs one at a time, and you will need to handle the four boundary cases before you have read the first five inputs.

4. *Making sense of data*: Convert the acquired readings from voltage format to temperature format in Celsius and Fahrenheit (C and F). The equations are to be found under the ADC section in the STM32F4xx datasheet.

5. *Temperature display:* You are required to display the temperature on a four digit 7-segment display in the form of *XX.Y*. To this end, you have to build the circuit given all the components you have acquired. All 7-segment data lines must share the same processor port/pins to save I/O pins. As such, to avoid conflict, you are instructed to make use of 7-segment multiplexing technique. Check the Appendix at the end of this experiment. You are required to read the blue button connected to PINA.0 for switching between Celsius and Fahrenheit whenever the button is pressed.

6. *ADC with DMA:* All the above mentioned parts have 7.5 score out of 8. You are required to configure ADC with Direct Memory Access (DMA), instead of simple mode, to get 8.5 out of 8. In this case, the ADC conversion runs in background and stores data into a location in the memory (DMA with circular mode). So, you only need to read the content of the location at the sampling frequency (mentioned in number 2).

Note the following:

- The 7 segment switching when done at a quite fast speed will fool the eyes to see the whole set of digits as being all on at the same time. You have to try different values of time delays to find which will give a consistent output. That is, no flickering or any obvious transitions should be visible.
- You can use SysTick timer for this purpose or software delays.
- There should be enough delay time for the current to actually turn the 7-segments LEDs on. The multiplexing procedure, if done right, yet with shorter than needed delays will not light on the displays.
- You will use common cathode displays and others common anode ones that need logic high to light and uses NPN transistors as a switch.
- **Don't forget to use the current limiting resistors** (Rs resistors, ranges in Ohm) with the display (either during testing them or using them). Otherwise, you risk damaging the display.
- You need to write some code to translate in between actual numbers and their 7-segment representations.
- You might choose to update the display at a lower rate than the sampling rate. That is, you might not need to write each of the acquired 25 values but only one of every five or ten. Writing at higher speeds might make reading the least significant digit more troublesome at it more likely to change at a faster rate.
- All of you got more resistors and transistors than you need in case you break some pins or lose some.

7. *Overheating alarm:* Finally, you have to set up an alarm mechanism that triggers when the temperature goes beyond a certain threshold. During the demo, we will heat the board by using external sources up to 60℃. So start by choosing a suitable threshold at which the alarm mechanism will kick off. The board has four LEDs that are placed in a plus sign shape. The alarm mechanism is simply having the four board LEDs turn ON in circular motion and in succession. The first on-board LED will turn ON for a while then turn OFF. The next LED will do the same and so on. Therefore, this LED succession will make the LEDs look like as if they are rotating. The alarm should be stopped when the temperature goes back into the safe range. Let the duration of the ON time of the LED be an appropriate multiple of the SysTick timer so that the LED will turn ON or OFF after an X number of SysTick interrupts.
   Hint: you could choose a fixed period time of your own through experimenting. Or you could use the SysTick 100Hz frequency as your base period.

Testing

In lab conditions and room temperature, most processor temperature readings should be below or around 30 C°. But these sensors have lots of error and values above 30 are reasonable to see. To force the processor to heat up, you might try a 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. **The best heating source is actually a hair dryer and this is the one the TAs will use during the demos.** Do not, however; subject the board to heat sources which could damage it. Don't even use ice to cool it down ☺

Hints

LEDs and Push buttons

The use of general-purpose IOs (GPIOs) and ADC API is fully described in "Doc_19 - Documentation of STM32F4xx Cube HAL drivers". To see how the LEDs and the button are connected to the processor, please consult "Doc_10 - Discovery Kit F4 Rev.C". You can listen to the push button by reading the value of the pin (polling) or enabling the related interrupt.

SysTick handlers and 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 user flag inside the ISR which can be observed from your regular application code once you return from your interrupt routine.

Reading and Reference Material

- Tutorial II by Ashraf Suyyagh
- C Tutorial by Ben Nahill
- Doc_03 - Technical and Electrical
- Doc_05 - STM32F4xxx Reference Manual
- Doc_10 - Discovery Kit F4 Rev.C
- Doc_16 - Debugging with Keil
- Doc_17 - Introduction to Keil 5.xx
- Doc_19 - Documentation of STM32F4xx Cube HAL drivers
- Doc_24 - Doxygen Manual 1.8.2 (Optional)
- Doc_25 - ProGit 2nd Edition (Optional)
- Doc_26 - General Report Grading Guideline for TAs Ver 1.3 (Report related)
- Doc_27 - Lab Report Guidelines ver 2.0 (Report related)
- Doc_31 - Clock Display Datasheet
- Of course you are free to refer to any other posted documents which you see useful.

Bonuses

1. **Early demo bonuses (13th – 16th February)**: Thursday demos have a bonus of 0.25, Wednesday's 0.375, Tuesday's 0.5 and Monday demos 0.625.

Demonstration

Demos will take place Friday, **February**, the 17th. We expect that your code be ready by your appointment time that day and that you will be ready to demo. No extensions allowed without penalties. You will lose 35% for the Monday (20th) demo (add 10% per day beyond Monday). Reports are due on Monday's midnight, late submissions will incur a penalty of 10% per day for the lab report. We take deadlines seriously. The experiments have been designed such as a group of two could handle it in less than two weeks' time **provided that they attend the lectures and Tutorials.**

Final Report Submission

Reports are due by Monday, February 20th, at 11:59 P.M. Late submission will be penalized by 10% a day. **Submit the PDF report and the compressed project file separately**. **Don't include your report in the archive. The ZIP file and PF names should be LAB2_Gxx.**

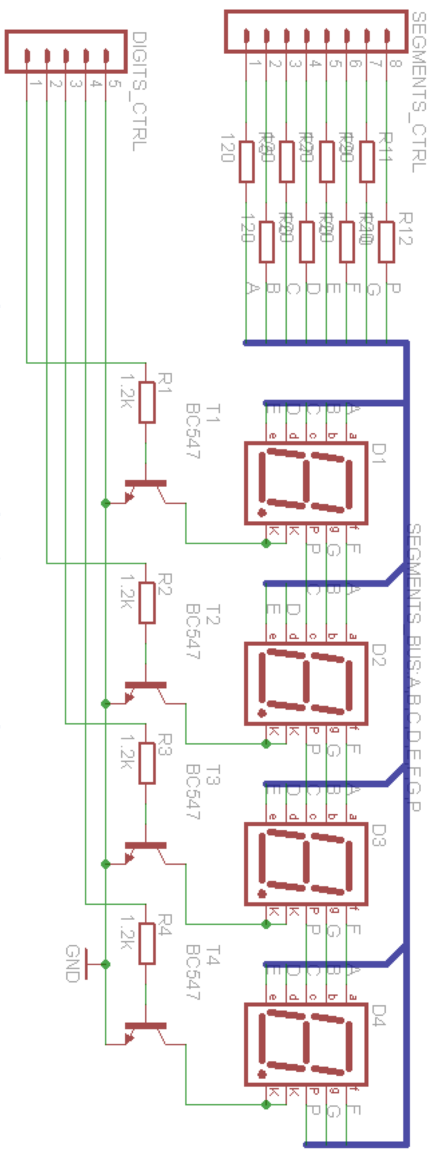

Figure 1- An example of 7Seg connection.