ECSE426 Microprocessor Systems

Fall 2016

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

Introduction

A common operation in embedded microprocessor systems is sensor data acquisition and signal processing. If the signal is known to be noisy, it is beneficial to apply a filter to remove some of that noise. In this experiment, you will construct such a system based on embedded temperature sensors. You will perform data acquisition and filtering while providing a simple graphical output using the STM32F4-Discovery's LEDs. You will also construct an external circuitry of 7-segment displays and interface it through the digital I/O pins to display the temperature numerically.

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 are ready for you to pick up. 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 code the same way you did in Lab 1 except for a few changes in the project setup. You will 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 myCourses for the executable). 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.

The tutorial covers in detail how to use peripheral drivers based on the STM32F4 Cube HAL firmware. There is also discussion concerning debugging, reading the board schematics and basics of interfacing. To learn more about embedded C programming, some slides as well as an embedded C tutorial have been posted for your reference.

The Experiment - Functional Requirements

In this lab you are required to do the following:

1. *Set up the 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 the processor speed or employ fans to cool the processor down. Although we won't be using the temperature sensor in such an 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. The sensor output is an analogue voltage, as is the case for many sensors. The sensor data should be acquired at a frequency of 100Hz. You are referred to the ADC section in the STM32F4xx datasheet. Also consult the tutorial and the STM32F4 Cube Documentation to get started with the setup. Detailed example code will be provided in the base project.

2. *Set 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: (i) setting up a required frequency and (ii) writing an associated interrupt handler (see stm32f4xx_it.c). The SysTick body should be lightweight. You are encouraged to investigate the driver code of the function for setting up the frequency. This will help you understand the timer's limitations (min and max supported frequencies).

3. *Filter the signal:* The digitized temperature sensor data is prone to noise. There are many sources of noise, including electromagnetic interference, thermal noise, and quantization noise due to the ADC. As such, one needs to filter these noise sources and minimize their effects. There are many candidate filters. To keep things simple, we will use the Kalman filter from lab 1 to do the filtering. In this case, we will use the simple model:

   $$z_k = x_k + v_k$$
   $$x_k = x_{k-1} + w_k$$

   Here z_k is the measured data at timestep k, x_k is the true underlying temperature and v_k and w_k are Gaussian noise terms. By choosing the values of the variances of the noise terms you can control (i) how quickly the state variable (true temperature) can change over time and (ii) how much of an impact new data has on the estimate.

   You are free to use either your ***assembly*** or ***C*** implementation of the filter. Make sure you have a correctly functioning filter. To test the filter, you can generate a test vector of your own in MATLAB and compare your filter output to MATLAB's. MATLAB is installed on the lab machines as well as various machines in the Trottier building.

   You are required to investigate which values of the filter parameters give good results. You should conduct an analysis of the temperature sensor output. The printf statement could be beneficial in extracting the data and pasting it into analysis software (e.g., Excel, Matlab, Python, R). Then, use any mathematical package of your choice to analyze and compare the original noisy data with different versions of the filtered data with various parameters. You have to present a convincing case to justify your final choice. Note that this is an engineering trade-off; there is no correct answer. By choosing small values for the variance of w_k, your estimate will be more robust to noisy data, but it will take longer to detect a genuine change in temperature. Conversely, larger values will allow you to detect rapid changes, but will make the estimate much noisier. You can form an estimate of an appropriate value for the measurement variance by trying to keep the temperature as stable as possible and taking many readings (maybe sampled at a slightly faster rate than we ask in the lab). You can then try to implement a change between two temperature levels by applying a heat source so that you can estimate how fast the temperature can reasonably vary (allowing you to choose an appropriate variance for w_k in the dynamic model).

We expect some visual analysis (i.e. graphs) of raw sensor data against filtered data with varying parameters.

4. *Convert the data*: Convert the acquired readings from voltage format to temperature format in Celsius. The equations can 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 the 7-segment multiplexing technique. See the diagram at the end of the experiment description.

Note the following:

- If the switching (displaying one digit then another) is performed at high speed, human eyes see all digits as been all on at the same time. You have to try different values of time delays to find which will give a consistent output. No flickering or obvious transitions should be visible
- You can use the SysTick timer for this purpose or software (or Cube HAL) delays.
- There should be enough delay time for the current to actually turn the 7-segments LEDs on. If the delays are too short, there will be no light on the displays, even if the multiplexing procedure is implemented correctly.
- Some of you will use common cathode displays and others will use common anode displays that need logic high to light and uses NPN transistors as a switch. Check to make sure you know which display you have.
- Don't forget to use the current limiting resistors (Rs resistors) with the display (either during testing them or using them). Otherwise, you risk damaging the display.
- You need to write code to translate 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 out 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 (more prone to change due to noise). Yet, even if you lower the display rate, make sure it is still responsive to quick temperature swings.

- All of you are provided with more resistors and transistors than you need in case you break some pins or lose some. (Please try not to do this too often, but accidents will happen).

6. *Implement an 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 for the alarm mechanism. 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 appear 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. You could choose a fixed period time of your own through experimentation. Or you could use the SysTick 100Hz frequency as your base period.

Testing

In lab conditions and at room temperature, most processor temperature readings should be below or around 30 C°. But these sensors have substantial 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 probably use during the demos. Please do not subject the board to heat sources that could damage it. Please don't use ice to cool it down ☺.

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 learn all about the advanced tools Keil provides for debugging, check the document **"Doc_16 - Debugging with Keil"**.

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 are connected to the processor, please consult "Doc_10 - Discovery Kit F4 Rev.C".

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.

Don't worry about interrupts outside of the SysTick timer. You will write a handler routine "void SysTick_Handler(void){}" which will execute each period of the SysTick timer. You are not required to use any **other** sources interrupt beside SysTick's of for this experiment.

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

Bonuses

1. You will be provided with piezo-electric sensors. If you implement code that can read from these sensors and perform ADC conversion, you will receive 0.5 bonus marks. If you can then turn on board lights that reflect the strength of the strike (more lights for hard strikes, less for softer strikes), you will receive another 0.5 bonus marks. If you do this, then you should change your overheating alarm for the temperature sensor to be a flashing of the 7-segment display.

Demonstration

Two demos will be held for this lab. The exact time will be decided later but we will post appointment slots for you to reserve. We expect that your code to be ready by your appointment time that day and that you will be ready to demo. No extensions will be allowed without penalties. You will lose 35% for a Monday demo (add 10% per day beyond Monday). There are bonuses for early demos – the TAs will announce these.

Demo 1: Friday Oct. 7

1) Set up the Systick timer to 1ms.
2) Demonstrate that you can control the on-board LEDs.
3) Build the 7-Segment display and show its ability to display numbers.
4) Set up ADC(s) for temperature sensor (and piezo sensor); show raw values being read.

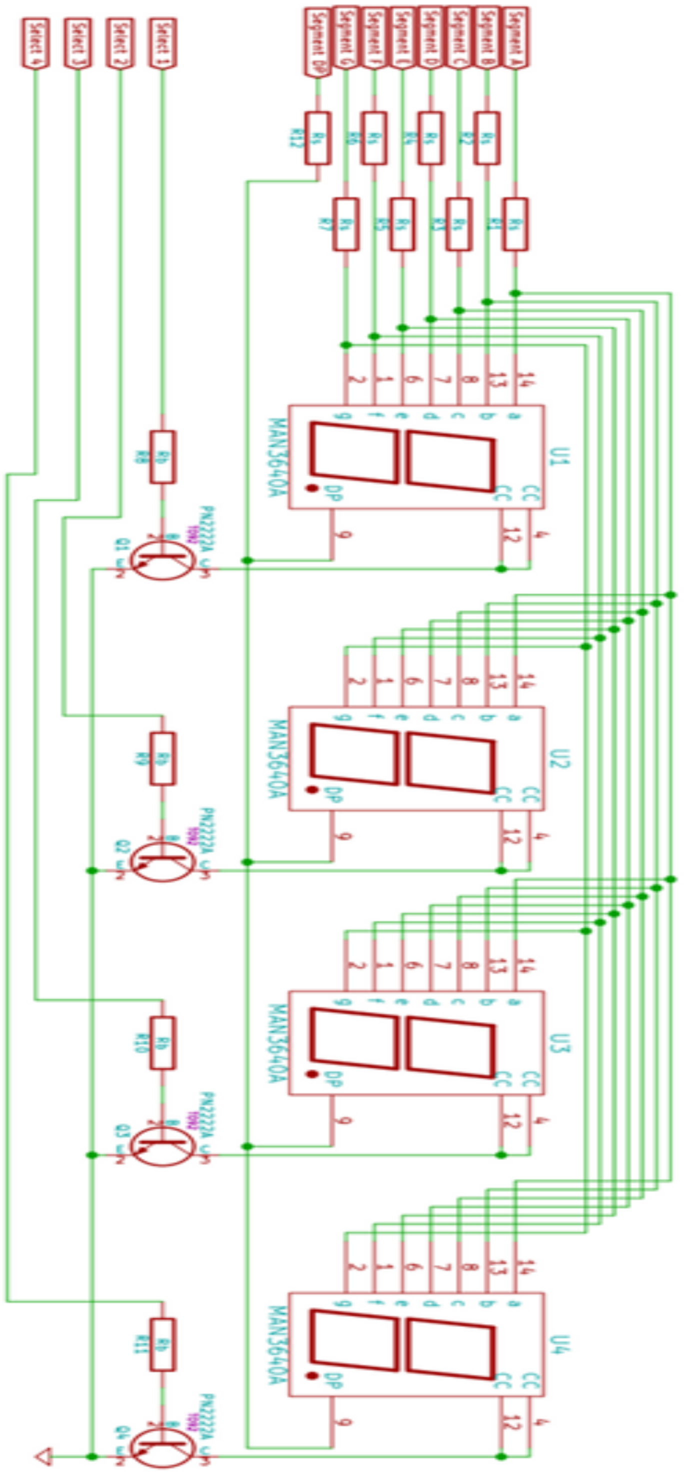
Demo 2: Friday Oct. 14

1) Poll values from the ADC(s) at 100 Hz; do the necessary calculations for conversion.
2) Display temperature on the 7-segment display.
3) Bonus: For the piezo sensor, turn on more board LEDs for harder strikes, fewer for softer ones.

Final Report Submission

Reports are due by Monday, Oct. 17, 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.**

Common-cathode 4-digit 7-segment interfacing


Select 4
Select 3
Select 2
Select 1
Segment DP
Segment G
Segment F
Segment E
Segment D
Segment C
Segment B
Segment A
U1
U2
U3
U4
MAN3640A
PN2222A
Q1
Q2
Q3
Q4