EENG260 (Microcontrollers) Project:

Build a simplified Sulfur Hexafluoride (SF_6) gas density monitor using provided hardware with a Tiva C evaluation board

Using the provided hardware modules, introduced in your previous labs, create a simplified SF_6 gas density monitor similar to those used by electric utilities worldwide.

Reference Materials

- CortexM_InstructionSet.pdf
- Tiva TM4C123GH6PM Microcontroller spms376e.pdf
- Tiva C Series TM4C123G LaunchPad spmu296.pdf
- LEDs and Pots.pdf
- Display Module.pdf
- TivaWare_C_Series-2.2.0.295.zip
- TivaWare Peripheral Driver Library spmu298e.pdf

What is SF_6, and why does it need monitoring?

A gas at room temperature and atmospheric pressure, SF_6 is a chemical with a number of interesting properties. Due to its molecular structure, it is an excellent insulator, providing a dielectric strength roughly 2.5 times that of air. In high-voltage circuit breakers used by electric utilities, where voltages can be in the tens or hundreds of thousands of volts, surrounding the breaker contacts with SF_6 allows for arcs to be quenched with less distance between the contacts, allowing for the design of smaller circuit breakers than would otherwise be possible. Due to this benefit, SF_6 circuit breakers have been used by electric utilities since at least the 1950s.

Of course, if a circuit breaker is built with an expectation of having a certain amount of SF_6 present when it's operated, having less than that amount could be a very bad thing, resulting in damage to the breaker and, possibly, failure of the breaker to interrupt current as intended. Knowing there is enough SF_6 on hand in a breaker is a key safety feature, making active monitoring worthwhile. However, many of these circuit breakers are in switchgear yards in isolated, sometimes inhospitable places, and it's not always feasible to have someone go check the breaker prior to needing to operate it, so automated monitoring is called for.

That would be enough on its own, but, in recent years, another issue has become apparent. At one time, if a breaker had a slow leak, the utility might have just topped off its SF_6 if it was less expensive to do so than to find and fix the leak. Unfortunately, per the Environmental Protection Agency, SF_6 has also been discovered to be the most potent greenhouse gas known to date, 22,800 times more effective than CO_2 at trapping infrared radiation, with an atmospheric lifetime of 3,200 years. SF_6 is a useful gas, but leaks need to be found and fixed in short order.

How do we monitor SF_6?

Since SF_6 is a colorless, odorless, chemically-stable, non-toxic gas, being used in some large pieces of equipment, we can't readily monitor for its presence by simple methods like measuring a fluid level or

by its weight. What we *can* do is rely on the fact that, if we fill a container with *only* SF_6 to the point where there is a positive pressure compared to the outside of the container, then nothing else will find its way inside (any leaks will solely result in the release of SF_6 to the surroundings). With that setup, we can rely on the Ideal Gas Law you may have been exposed to in a previous Chemistry class to track what's going on inside the container:

$PV = nRT$

…where P is the pressure (on a scale where zero is a vacuum), V is the volume of the container, n is the amount of gas by molecular count (directly related to the mass of the gas), T is the temperature (on a scale where zero is absolute zero), and R is a constant chosen to make all the units work out. If you have a constant amount of gas present, and you have a rigid container (so the volume is constant), you can collapse all those constants into a simple scaling factor (let's call it s), making the new equation:

$P = sT$

…meaning that, for any given change in temperature, you can just multiply by that scaling factor to work out what the pressure *should* be. So, if you constantly measure the pressure and temperature of the gas, you can get what the pressure is vs. what the pressure should be, both of which are directly proportional to how much gas is (or should be) present (if you go back to the Ideal Gas Law above, if volume and temperature are fixed, the only thing changing the pressure is the amount of gas). So, actual pressure can then be calculated and displayed as a percentage of "should be" pressure, equivalent to the percent of full density of gas present, to tell the utility personnel whether they have as much SF_6 as they need, or they think they should have. All we need is the means to measure the temperature and pressure, and we're good to go.

I should point out here, all of this depends on the Ideal Gas Law being correct. In truth, one of the interesting properties of SF_6 is that it doesn't quite follow the expectations of the Ideal Gas Law exactly. However, those deviations are calculable, so the overall concept still holds, while it doesn't add anything useful to this project, so we will politely ignore that issue, as part of building a *simplified* monitor.

How do we measure temperature and pressure?

There are a variety of sensor types available for measuring temperature, pressure, and other real-world phenomena, each relying on a different method to produce a measurable output. Commonly, this output comes in the form of an analog voltage or current that varies linearly in direct relation to the thing being measured. For example, a pressure sensor might cover a range of 0-100psi while producing an output from 0-5V DC, meaning it would indicate a 20psi pressure with a 1V output.

Since we're dealing with analog values, we need to use an Analog to Digital Converter (ADC) to convert the sensor data into a format useable by our microcontroller. In later courses, you might be asked to either select sensors suitable for your needs, or to design circuitry to interface sensors with an ADC, accounting for things like the maximum voltage allowed by your ADC or noise induced from external sources. For this project, we will just use a pair of potentiometers hooked up to provide voltage to imitate sensor outputs.

The ADCs on our microcontroller cover a range of 0-3.3V, which is converted into a range of 12-bit integers, from 0 at 0V to 4095 (0xFFF) at 3.3V, with each step on that scale (the ADC's "resolution")

being equivalent to a change of just over 0.8mV. Again, in a more advanced course, you might be asked to evaluate whether the ADC is suitable to the task you're asking it to perform, but, for this project, you can assume that this resolution is acceptable.

All that remains is how to configure our microcontroller's ADCs to measure the voltages our sensors are providing. Our microcontroller has a pair of ADCs that can "sample" any of 12 different shared input lines, which are alternate functions of some of the GPIO pins. The pressure sensor will be on analog input 11 (port B pin 5), and the temperature sensor will be on analog input 8 (Port E pin 5). Since we have two ADCs to work with (and the electric utilities won't care about the extra power draw), you can configure one ADC solely for pressure and the other solely for temperature, with interrupts used to update your sensed temperature and pressure.

Of course, appropriately for the final project for the class, the ADC modules on our microprocessor are unusually complex, with a larger variety of configuration options than ever before. Again, don't get distracted by things outside of your main goal here. All you want to do is configure the appropriate GPIO pins to pass the analog signals you need to the ADCs, then configure the ADCs to generate an interrupt when they've got your data, then retrieve the data and start another sample cycle. Beyond the obvious, here are a few items to help you along:

- The ADC modules require a 16MHz clock to operate (good thing you're using a 16MHZ crystal clock source, right?).
- Since the data we're tracking changes extremely slowly, we prefer accuracy over speed of data acquisition, so maximum *hardware oversampling* (letting the ADC do several samples, then averaging out the results before returning a value) should be used.
- Since we're not giving the ADC's any sort of break between samples, you should be sure to set the DITHER bit to reduce random noise, per section 13.3.2.8 of the microcontroller manual.
- The ADC modules are built around a concept of "sample sequences", giving each module a series of measurements to complete before it generates an interrupt saying it's completed all those tasks. So, there is no simple "just take a measurement and get back to me when you're done" option. Rather, you have to configure the ADC to use the proper sample sequencer (SS3, which does one sample and prepares one value), then configure that sequencer to use its one sequencer step to take the measurement and generate the interrupt.
- When the time comes to retrieve the data your ADC has generated, the necessary function has been written to use a *pointer* for one of its parameters, to allow the value(s) generated by the ADC to be *passed by reference*. These are concepts that should have been covered in your previous C class, but I've heard that time constraints may have caused this section to be dropped by some instructors in the past. If you are in this position, let me know, and I will do a brief lecture to cover this topic. **If you don't know how to use a pointer to pass data by reference, you will not be able to get data from your ADC, and you will be unable to complete this project.**

On to the project

Here are the hardware specifications for your device, given to you by your employer:

- A 3-digit, 7-segment display, identical to what you used in Lab 5 (with data lines on Port D pins 0-3 and control lines on Port E pins 1-3)

- A display mode indicator (represented by the LEDs on the microcontroller evaluation board)
- An alarm indicator (represented by an LED on the LEDs and Potentiometers module, connected to Port D pin 6)
- A display mode selector switch (SW1 on the microcontroller evaluation board)
- A pressure sensor (represented by one of the potentiometers on the LEDs and Potentiometers module) connected to Analog Input 11
    - Sensor measures 0-100 psia (pounds per square inch absolute, meaning 0 is vacuum, not atmospheric pressure)
    - Sensor produces 0V at 0psia, 3.3V at 100psia, and changes linearly across its range (so, for example, at 50psia, the sensor would produce 1.65V.
- A temperature sensor (represented by the other potentiometer on the LEDs and Potentiometers module) connected to Analog Input 8
    - Sensor measures -40C to 60C (Celsius)
    - Sensor produces 0V at -40C, 3.3V at 60C, and changes linearly across its range.
- The manufacturer of the circuit breaker this monitor will be attached to has declared that a full load of SF_6 will produce a pressure of 50psia at 20C
    - Converting the temperature to Kelvin (a temperature measurement scale where 0 is absolute zero, and handily converts from Celsius (K = C + 273)) results in an expected pressure scaling factor of 0.17065psia per degree K.

Here is what your software needs to make that hardware do:

- Constantly monitor temperature and pressure
- Calculate expected pressure from the temperature data
- Calculate percent density (actual pressure vs. expected pressure)
- These are the rules for the seven-segment displays and display mode indicator:
    - The seven-segment displays will show the integer portion of whatever value is to be displayed by the currently selected mode
    - Density mode will show the density percentage (100% shows as 100) on the seven-segment displays, and the mode indicator will show green
    - Temperature mode will show the temperature on a custom scale (0 for -40C, 100 for 60C) on the seven-segment displays, and the mode indicator will show red
    - Pressure mode will show the pressure in psia on the seven-segment displays, and the mode indicator will show blue
- At startup, the display mode will be Density.
- Pressing the display mode selector switch will change the display mode from Density to Pressure, Pressure to Temperature, or Temperature to Density.
- If the current display mode is anything other than Density, and it has been 5 seconds since the last time the mode selector switch was pressed, the display will automatically change to Density mode.
- These are the rules for the alarm indicator:
    - If the density drops below 85% while the alarm is off, the alarm will turn on.
    - If the density rises above 90% while the alarm is on, the alarm will turn off.

Management may add further requirements up to two weeks before this project is due.

To turn in

Make a .zip file with the following contents and upload it to Canvas:

- A brief report, in Word or PDF format. What issues did you run into? What did you learn? Keep it under a page, if you can.
- A copy of your main.c file. You can export your file to the file system by right-clicking on it in the Project Explorer pane and selecting Export.
- Any other .c and .h files you added to your project (if any)