ECE381 – Lab 7 – Analog Inputs & Outputs

DUE: 11 April 2023

Introduction:

Many sensors output analog values, and if suitable controllers are to be developed, Analog-to-Digital Converters (ADCs) will be used to quantize these analog values into digital for control. Most modern microcontrollers will have ADCs built into the hardware. To complement ADCs, many microcontrollers also have the complement, Digital-to-Analog Converters (DAC), which can produce an analog output given an input digital value. In this lab, we will use both. We will use a DAC and ADC to calibrate our analog values. Then, we will use this calibration to read and display a temperature from an analog temperature sensor, or use it to have an analog controlled LED sequencer.

Background:

Analog-to-Digital Converters(ADCs)

Analog-to-Digital Converters (ADCs) are analog devices which take as input an analog voltage and produce as output an digital value. There are three main characteristics of ADCs, the input *range* of voltages, the *resolution* in bits, and the *speed* (often called the sample rate, expressed in samples per second or conversions per second). The range defines the lowest possible voltage (0V typically for an unsigned ADC, negative rail for a signed ADC) to the highest possible voltage based on the resolution, in bits, of the ADC. For example, the ADC could be set from GND to Vdd, (0V to 5V). If we configure it for 8-bit resolution with this range, a sampled digital value of 0x00 would be GND, a sampled value of 0xFF would be VDD, and a sample value of 0x80 would be 2.5V. In general the relationship is:

$$\frac{V_{max}-V_{min}}{2^{resolution}-1}=\frac{5\,V-0\,V}{2^{8}-1}=\frac{5}{255}\rightarrow 19.6\frac{mV}{Digital\,Value}$$

Since an ADC requires time to sample, the sample rate of an ADC limits how quickly the input voltage value can change without causing distortion or aliasing. The Nyquist limit states that your ADC must sample at least twice the highest expected frequency. Different ADC types do the conversion in different ways. The PSoC4 has a single hardware Sequencing Successive-Approximation (SAR) ADC, which uses a Digital-to-Analog Converter (DAC) and comparator to do a sequence of > comparisons to arrive at the final value like this: analog_value > DAC (Vdd/2 = 2.5) → (MSB=1 else MSB=0), (If MSB=0, DAC set to 1.25 for next comparison otherwise DAC set to 3.75 for next, etc. You are basically doing a binary search through the voltages). The sample rate and resolution of a SAR ADC are based on the speed of the DAC, speed of comparator, and resolution of the DAC.

Another aspect of ADCs is how they are referenced. A *ratiometric* ADC will typically sample the voltage as a ratio of the max./min. voltage of the input range (ie. sampling the middle of a resistive divider between VDD and GND). An *absolute* ADC will use a defined voltage reference (ie. a bandgap reference or other absolute, constant voltage value) when generating digital values. Different scenarios require different configurations. For example, a thermistor whose resistance changes with temperature would require ratiometric mode to determine the temperature, but a sensor like our MCP9701 produces exactly 19.5 mV/°C and thus should use an absolute reference for accurate measurement.

Digital-to-Analog Converters (DACs)

Digital-to-Analog Converters (DACs) are analog devices who take as input a digital value and produce as output an analog voltage or current. The three main characteristics of DACs are the same as ADCs; the output *range* of voltages, the *resolution* in bits, and the *speed* (often expressed in samples per second). The range defines the lowest possible voltage/current (0V/0A typically for an unsigned DAC, negative rail for a signed DAC) to the highest possible voltage/current based on the resolution, in bits, of the DAC.

For example, the IDAC we will use on the PSoC will be configured for 8-bit resolution with a range of 0V to 306 µA. This means that if you write the digital value 0x00 to the IDAC, it will nominally produce 0A, if you were to write the digital value 0xFF, you would get approximately 306 µA, and the IDAC would output voltage would vary linearly between these values at a rate of:

$$\frac{306\mu A - 0\mu A}{255 - 0} \rightarrow 1.2\frac{\mu A}{Digital\,Value}$$

So an input of digital value 0x80 would produce 153µA, 0x40 would produce 76.5µA, etc. Since voltages/currents cannot change instantaneously, the speed of a DAC limits how often the output sample value can change without causing distortion. The IDAC on our PSoC has a maximum settling time of 10 µsec, so that output limit is roughly 100 ksps (100 thousand changes per second).

The Temperature Sensor

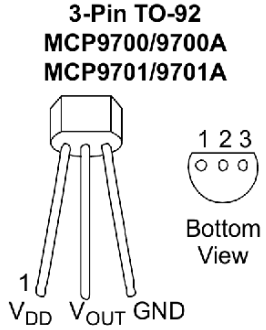


Figure 1: MCP9701 Sensor Pinout

The MCP9701 is an analog temperature sensor that produces a 19.5 mV/℃ change on its output node. The absolute temperature is measured using the equation given in the datasheet:
$V_{OUT} = T_C \times T_A + V_{0℃}$
where TC is the temperature coefficient (19.5 mV/℃), TA is the ambient temperature (what we want to measure), and V0℃ is the voltage at 0℃, which should be somewhere around 400mV according to the datasheet. Therefore, by knowing the output voltage by measuring with the ADC, we can compute the ambient temperature.

PART 1 - ADC/DAC Calibration

In the first part of the lab, we are going to calibrate our ADC so that we know exactly what DV output by the ADC corresponds to which absolute voltage. We will do so by using the DAC with a known resistance to set a voltage, then use the ADC to sample that voltage and see which DV results.

Basically, if you measure a known resistance with a multimeter, and you output a known current, you have a known voltage. But your ADC doesn't give you voltage, it gives you DVs. So you are fitting the known voltages to these DVs to make a line. For example, let's say you grab a 10K resistor, measure, and have a known, measured resistance of 9.98K. If you set the IDAC, as it's supposed to be configured below, to value 10 (IDAC_SetValue(10) in code), that should output 10x1.2uA = 12uA. 12uA times 9.98K would be the voltage 119.76 mV. When you call the ADC_GetResult16() function to get the DV, the voltage it is measuring should nominally be (2.048/4095) times that DV returned by ADC_GetResult16. That DV *should* be 0.11976V / (2.048V/4095) which is 239 (or 240 depending on quantization error). It might not be, so by fitting a bunch of known voltages to ADC values, you get a linear equation that maps ADC values to voltage. Basically in a spreadsheet, you would have two columns, again using the example here:

ADC Value	**Known Voltage**
ADC_GetResult16() when IDAC set to 0	0V
ADC_GetResult16() when IDAC set to 10	0.11976V
...	...

By using ADC Value as the X axis and the Known Voltage as the Y axis, you get an equation:

Real Voltage = coeff_a * *What_ADC_GetResult16()_Gives_in_ANY_Program* + coeff_b

Where coeff_a and coeff_b are given by the spreadsheet fit. You can use RealTerm's capture capability to save it in a format that Excel can parse out (I think just a comma in between DV and voltage with a new line per row for each DV).

PSoC Configuration:

- Place an “Analog/ADC/Sequencing SAR ADC”. Double or Right-click to bring up the Configuration window. Rename it to ADC. Configure the input range to be 0 to 2*Vref by using the “Internal 1.024 volts, bypassed” Vref option (though this requires the BYPASS cap to pin on 1[7] to be selected in the Pins window) and setting “Single ended negative input” to “Vref”. The ADC is differential, so by setting the negative terminal to Vref, you are allowing the + terminal to go +/- Vref around the negative, for a range of 0 to 2*Vref! Click on the “Channels” tab and input only 1 Sequencing Channel. Change the mode of the channel to be Single at 12-bits of resolution. Go back to the “General” tab and make sure the “Single ended result format” is set to Unsigned (so 0V maps to digital value 0)
- Place a “Ports and Pins/Analog Pin” and connect it to the input of the ADC. Rename it to ADC_IN. Map it to P2[7]. Port2 has direct connection to the SAR mux inputs, so it’s best to place analog inputs on this port. You can place them on other ports, but then they will have to be routed to the ADC, potentially adding noise to the input path.
- Place an “Analog/DAC/Current DAC”. Configure the polarity to “Positive (Source)”, the resolution to “8-bit”, and the Range to “0-306 uA (1.2 uA/bit)”
- Place a “Ports and Pins/Analog Pin”. Connect it to the output of the IDAC. You might want to right-click on the pin, and select “Shape → Flip Horizontal” to put the pin on the left side for easy connection. You can apparently map the IDAC to any pin you’d like. I’d choose P0[2] or P0[3].
- Place a “Communications/UART (SCB Mode)”. Change the baud rate to 57600. Map RX & TX to P7[0] and P7[1] respectively.

Software:

- Connect a resistor from P0[2] (or whatever pin you used for your IDAC) to GND. Measure the value of the resistor with a multi-meter. Ensure the resistor is large enough to provide good measurements over the ADC range knowing the IDAC range (0 - 306 µA) given the range of your ADC
- To calibrate the sensor, drive it with a few known currents with the IDAC. Include 0x00, 0xFF, and at least 5 other values in between. If you have the UART formatting figured out for easy Excel import, there really is no reason to not do all 256 values
- Import the data in Excel, do the fit of your DV to the voltage, and note the coefficients for Part 2.

PART 2 - Temperature Sensor/Analog LED Controller

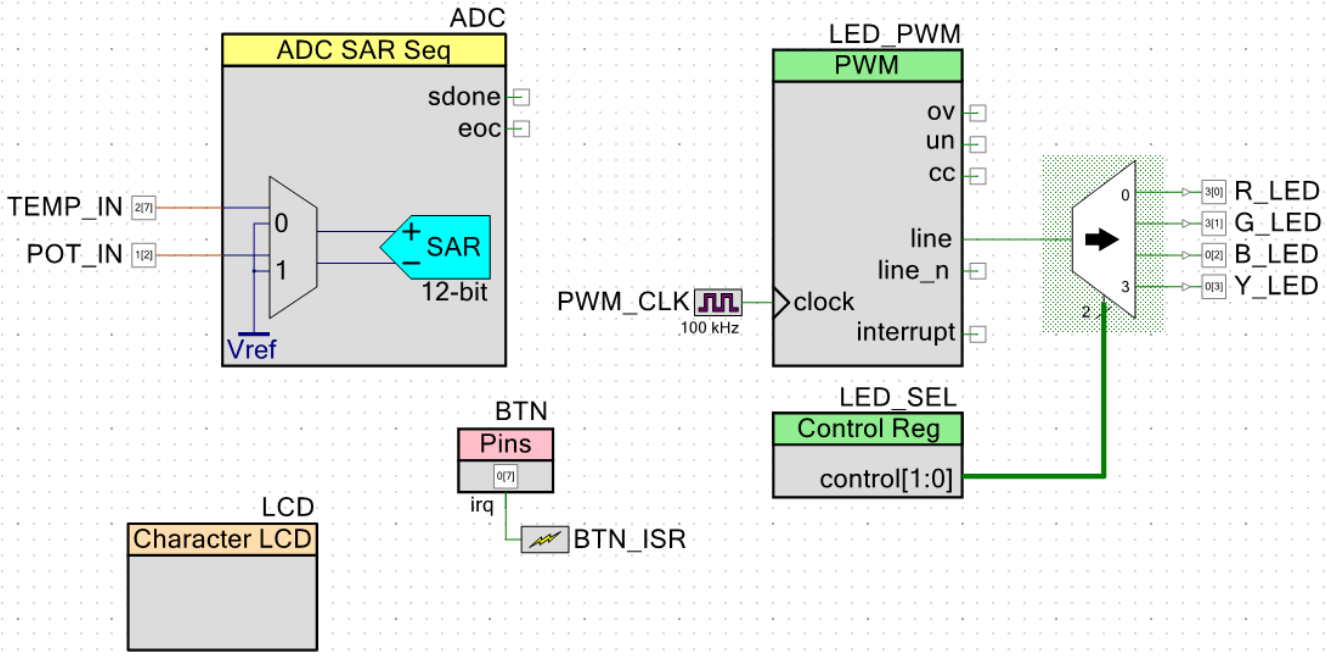
In part 2, we are going to have to analog inputs, an LCD, and a stepper motor for output. The user will pick one of two modes. In temperature mode, the user will sample the MCP9701 sensor, convert the temperature, and display on the LCD using the ADC in absolute mode (with Vref = 1.024 internal

reference). The R/G/B/Y LEDs and their brightness will be controlled by the current temperature value using a range specified below. By pressing the switch on P0[7], the program will sample the voltage on a potentiometer, then map this voltage to the R/G/B/Y LEDs and their brightness using a range below. The LCD will display the voltage of the ADC continually sampling the potentiometer input for changes. Since it will be sampling a potentiometer, the ADC reference will be in ratiometric mode (with Vref = VDDA/2).

PSoC Configuration:

- Place an "Analog/ADC/Sequencing SAR ADC". Double or Right-click to bring up the Configuration window. Rename it to ADC. Configure the input range to be 0 to 2*Vref by using the "Internal 1.024 volts, bypassed" Vref option (though this requires the BYPASS cap to pin on 1[7] to be selected in the Pins window) and setting "Single ended negative input" to "Vref". The ADC is differential, so by setting the negative terminal to Vref, you are allowing the + terminal to go +/- Vref around the negative, for a range of 0 to 2*Vref! Click on the "Channels" tab and input ensure 2 Sequencing Channels. Change the mode of the channels to be Single at 12-bits of resolution. Go back to the "General" tab and make sure the "Single ended result format" is set to Unsigned (so 0V maps to digital value 0)
- Place a "Ports and Pins/Analog Pin" and connect it to input 0 of the Analog MUX to the ADC. Rename it to TEMP_IN. Map it to P2[7]. Port2 has direct connection to the SAR mux inputs, so it's best to place analog inputs on this port. You can place them on other ports, but then they will have to be routed to the ADC, potentially adding noise to the input path. Use P2[7] for the MCP9701 sensor input.
- Place a 2nd "Ports and Pins/Analog Pin" and connect it to input 0 of the Analog MUX to the ADC. Rename it to TEMP_IN. Map it to P1[2]. Use P1[2] for the potentiometer input.
- Place a Digital Input Pin, configure for Resistive Pull-Up and map to P0[7] named SW. You can attach an ISR or use polling, it's up to you.
- Place a "Display/Character LCD" and map to P2[6:0].
- Place a Digital/Functions/PWM (TCPWM mode) and name it PWM.
- Place a System/Clock, connect it to the PWM's clock input and name it PWM_CLK.
- Place a Digital/Logic/De-Multiplexer. Change the number of output terminals to 4. Connect the input to the *line* output of the PWM. Place four Ports and Pins/Digital Output Pins and connect each to the 4 outputs of the De-Mux, naming them R/G/B/Y_LED as in past labs. Map R_LED to P3[0], G_LED to P3[1], B_LED to P0[2], and Y_LED to P0[3].
- Place a Digital/Registers/Control Register. Change the number of outputs to 2, make them a bus, and connect them to the select line of the De-Mux. Name it LED_SEL.

- The final version of the PSoC Configuration should look something like this:



Software:

- On startup, the motor should be off. The ADC should be configured in absolute mode from above and should start in Temperature Mode.
- In Temperature Mode, sample the MCP9701, convert the ADC measurement to the temperature using your fit from Part 1, and display the temperature on the LCD in both Celsius and Fahrenheit
- The R/G/B/Y LEDs are driven by a PWM, whos output duty cycle should be scaled to the range of operation. For all temperatures 0 - 75F, the blue LED should be on with duty cycle based on the current temperature in that range, for 75F - 80F, the green LED should be on with duty cycle scaled on the temperature in that range (75 should be 0%, 80 should be 100%, 77.5 would be 50%), 80F - 85F should be Yellow, and 85F - 212F should be RED, all with proper duty cycle scaling in that range.
- If the user presses SW, the program should go into Voltage Mode which entails:
    - Set the ADC reference to VDD/2 by setting bits 6:4 in the ADC_SAR_CTRL_REG to 110 (6), ensuring bit 7 in the ADC_SAR_CTRL_REG is also set to 1 to keep the bypass capacitor enabled (pg. 200 PSoC4 Architecture TRM)
        - The ADC_SAR_CTRL_REG name can be used for memory-mapping the register, as it is defined in the SAR ADC's API that gets generated
    - Change the ADC input mux to input 1
    - Display the current voltage on the LCD
- In Voltage Mode, continually sample the center-tap of the potentiometer. The LEDs should be output and scaled (duty cycle at low end would be 0%, duty cycle halfway through range would be 50%, duty cycle at max would be 100%) based on the following ranges

    - 0V - 0.8V: Blue LED
    - 0.8V - 1.6V: Green LED
    - 1.6V - 2.4V: Yellow LED
    - 2.4 - 3.3V: Red LED
- If SW is pressed again, go back into Temperature mode, which entails:
    - Set the ADC reference to 1.024 absolute by setting bits 6:4 in the ADC_SAR_CTRL_REG to 100 (4), ensuring bit 7 in the ADC_SAR_CTRL_REG is also set to 1 to keep the bypass capacitor enabled (pg. 200 PSoC4 Architecture TRM)
    - Change the ADC input mux to input 0
    - Turn the motor off
- In either mode, you can have a little delay in your main loop to keep the LCD from over-refreshing

Requirements:

- A measured calibration curve for your ADC, with linear fit, generated by the IDAC as specified in Part 1
- A continually updating LCD temperature display in both C and F using the MCP9701 as input, calibrated for accuracy, with LED color and brightness output based on current temperature according to the scales
- A continually updating LCD voltage display, with LED color and brightness output based on current temperature according to the scales
- A button that can toggle between the two modes