Introduction to Mechatronics (ME/AE 6705)

Lab Assignment 5

Analog to Digital Conversion

Objective

The main objective of this lab is to learn to configure the MSPM0 Analog to Digital converter for continuous conversion of a temperature sensor reading. The lab makes use of the Driverlib library, although you are welcome to use direct register access instead if desired.

Deliverables and Grading

To get credit for this lab assignment you must:

1. Submit a typed report as a PDF file to Canvas, answering the questions at the end of the lab (2 pages max, can be shorter). This is due at the beginning of class on the due date specified on Canvas. (20 points)

2. Demonstrate proper operation of your code to TAs or instructor during the office or demo hours. You must write your own code for this lab – no lab groups are allowed. (40 points)

3. Submit the commented final version of your code on Canvas (.c files containing your main.c code, and other associated files if needed.) (Pass/Fail)

Setup

This lab requires Code Composer Studio, the MSPM0, and the temperature sensor circuit built in Lab 1. The lab uses onboard features of the MCU such as the ADC and UART. The MSPM0 has a user configurable ADC module known as ADC12. There are two ADC modules which can be used in parallel or individually, with each module containing 8 channels. Each channel has a GPIO pin assigned to it to accept analog signals from external hardware.

Note: To create a new CCS project for the MSPM0 target device, follow the instructions in the document entitled "How to create a new project for the MSPM0.pdf" on Canvas under MSPM0 Documents.

Problem Statement

Write a program that reads the analog signal given to a pin corresponding to one of the *ADC12* channels. This ADC will convert an analog signal into a digital signal consisting of *12 bits*. The ADC module must be configured to scale the input between *+VCC (3.3V)* and *GND (0V)*. Care must be taken to tune the sensor output so as to reduce the maximum output voltage to +3.3V. (Please note if this is violated, the ADC module may be damaged due to high current draw and may be unusable thereafter.)

Background

ADC12:

The MSPM0 has two Analog to Digital conversion modules with eight channels in each module. Each module can be configured to provide 8 bit, 10 bit, or 12 bit outputs. Using more bits will provide better resolution (i.e. smaller voltage increments). But using more bits also increases conversion time because the MSPM0 uses a successive approximation ADC. More details on the number of clock cycles required for conversion can be found in the datasheet.

The first step in configuring the ADC is to determine the number of bits to be used. In this lab we will use 12 bits. This is the default so no additional configuration is necessary. Second, a sampling mode must be determined. ADC12 offers two different modes of conversion – manual and automatic. In manual mode, the sample is commanded by setting the trigger bit (SC) high and manually setting it low again. In auto mode, the sample is commanded by setting the trigger bit (SC) high, and then it automatically goes low after a certain specified sampling time. The one used in this lab is auto trigger mode. More details on these modes can be found in the reference manual.

Memory should be allocated to save results of the conversion. A function provided in the driver library (`DL_ADC12_configConversionMem`) can be used to configure memory. This function also sets the range of voltages to be used, the channel to be selected, and other parameters. In this lab we will use the following options when configuring the ADC:

- `ADC0` (ADC module 0)
- `DL_ADC12_MEM_IDX_0` (ADC result register)
- `DL_ADC12_INPUT_CHAN_2` (ADC input channel 2)
- `DL_ADC12_REFERENCE_VOLTAGE_VDDA` (uses the 3.3V reference on pin VDDA as the reference voltage, meaning the ADC will have a range of 0-3.3V)
- `DL_ADC12_SAMPLE_TIMER_SOURCE_SCOMP0`
- `DL_ADC12_AVERAGING_MODE_DISABLED`
- `DL_ADC12_BURN_OUT_SOURCE_DISABLED`
- `DL_ADC12_TRIGGER_MODE_AUTO_NEXT`
- `DL_ADC12_WINDOWS_COMP_MODE_DISABLED`

In addition to configuring the conversion memory, the functions that must be called to configure and run the ADC are (call in the order shown below):

- `DL_ADC12_enablePower`: Powers the ADC peripheral
- `DL_ADC12_setClockConfig`: Configures the ADC clock. Use the ULP clock with a divider of 8, and `DL_ADC12_CLOCK_FREQ_RANGE_24_TO_32` to set the frequency range
- `DL_ADC12_configConversionMem`: Call as described above
- `DL_ADC12_setSampleTime0`: Sets the time for each sample
- `DL_ADC12_enableConversions`: Enables conversions
- `DL_ADC12_startConversion`: Triggers conversions
- `DL_ADC12_getStatus`: Returns a macro that tells if a conversion/sample is in progress

- `DL_ADC12_getMemResult`: Returns the conversion result for a specified memory channel

NOTE: You will have to call DL_ADC12_enableConversions() each time after a sample has been taken. So call DL_ADC12_enableConversions() and DL_ADC12_startConversion() each time you want a conversion to occur.

Please see the documentation for the specific argument list for each function above. After complete configuration, a conversion can be initiated by calls to DL_ADC12_enableConversions() and DL_ADC12_startConversion(). The `DL_ADC12_getStatus` function can be polled to determine if the conversion is in progress. Once complete, the ADC result can be read using the `DL_ADC12_getMemResult` function and the conversion trigger can be toggled again to initiate a new conversion.

Hardware:

The circuit assembled/built in Lab 1 will be interfaced with the MSPM0 in this lab. Adjust the potentiometer used to tune the gain of the op-amp to set the gain such that the output voltage goes only up to 3.3V for 150°F to be safe. Connect the output of the temperature sensor signal conditioning circuit to the pin corresponding to ADC module 0, channel 2.

Software:

The program should convert and display the current temperature at 1 sec intervals to the console display in CCS. The conversion should be triggered at 1 sec intervals using *TIMG*. Configure *TIMG* such that it produces an interrupt every 1 second, and place the ADC conversion trigger (both the DL_ADC12_enableConversions() and DL_ADC12_startConversion(), and the print statement) inside the interrupt service routine.

- After a conversion is complete, print the results to the console display using the `printf()` function. To use `printf()`, you must include the `<stdio.h>` header file. No additional configuration is needed beyond this to use `printf()`.

When printing, make sure to convert the ADC output to the actual temperature using the appropriate conversion constants.

Requirements

1. Successfully demonstrate the outcome of the program and all the required functionalities to TAs or instructor.

2. Submit the commented final version of the code on Canvas.

3. Answer the below questions and submit a typed report to Canvas.

Questions

1. Take data from the above setup for approximately 60 sec. Around 30 sec, heat up the temperature sensor (either by placing you fingers on it or through other means). On the plot, show the two steady-state temperatures achieved by the sensor. How long does it take for the sensor to increase from the first steady-state temperature to the second? (Note: You can copy and paste data from the CCS console window directly to the plotting program of your choice, for instance MATLAB or Excel).

2. Calculate the resolution used in this lab in units of mV and in corresponding units of temperature.

3. An ADC is configured with 14 bits output. The output is scaled to 0V to 5V corresponding to the angular velocity of a motor from 0 to 100 rad/s. Calculate the voltage input to the ADC and output of the ADC if the angular velocity of the machine is 65 rad/s.