Project 3

ADC, DMA and DSP

ECEN 5813, Fall 2018

Prof. Kevin Gross

# Introduction

In this project you will set up a channel of the KL25Z onboard analog-to-digital converter (ADC) for continuous sampling of an input signal. Sampled data will be moved from the ADC to memory using the direct memory access (DMA) controller. You will be using a double-buffered data structure to manage the data. An application you write will examine the received data and perform analysis and print real-time information about the input signal out the UART.

# Guidelines

* Since this project uses the UART to output results, you may choose to base your project on your Project 1 port to KDS, Project 2 or start from scratch.
* Use the Kinetis board support package
* Follow ESE coding guidelines
* Have boundary checks on arguments and NULL pointer checks if required
* Use MACROS instead of hard coding other values. Macro names should begin with module name. For example, DMA\_xx, ADC\_xx.
* Work should be committed to the repository throughout development but at a minimum after completion of each part as outlined below.
* Students may work in groups. Groups may share information but each group must do their own work on the project.

# Part 1: Block diagram

In this project we will be receiving continuous data from the onboard ADC into double-buffered memory. An interrupt triggers data processing by the application and results of the analysis are outputted to the UART. Draw a block diagram featuring these elements and indicate the flow of data through them.

# Part 2: ADC setup

Configure processor pins 13 and 14 for use with differential-mode channel 0 of the ADC. This signal is available at pins 1 and 3 of J10 on the FRDM development board. Configure the ADC to continuously sample channel 0 with 16-bit resolution. Sampling frequency is determined by the clock provided to the ADC, the programmed divide factor, channel and sampling mode. The default ADC clock is the 48 MHz bus clock. A sampling frequency of 8 kHz or higher should be adequate for this project.

An ADC needs reference voltages. On FRDM they are pre-connected to the 3.3v supply and ground. Signals applied should not exceed the 3.3v power supply or go negative with respect to the FRDM's ground. **Damage may result if these limits are not respected.** A 10 kOhm series resistor and attention to ESD will provide some protection to the processor. See [this article](https://www.digikey.com/en/articles/techzone/2012/apr/protecting-inputs-in-digital-electronics) for additional background.

Do not enable any interrupt or DMA requests at this point. All data will be retrieved through polling in this part of the project.

Write a simple application that prints raw sample values to the UART. Note that ADC data is in the 16 least-significant bits of the ADC data result register (ADCx\_Rn). Results should be displayed as a signed number with range -32768 to 32767. Monitor the ADC digital output as you apply an analog signal to the ADC inputs.

## Questions

1. How is antialiasing handled in this system?
2. What is the behavior of the system when no input signal is applied?

# Part 3: DMA setup

In this part you set up a buffer for incoming ADC data and configure DMA channel 0 to continuously transfer samples from the ADC to the buffer. The buffer size should be a power of two and the buffer should start on a corresponding power-of-two address boundary. aligned\_alloc() can be used to create such a buffer dynamically. A total buffer size of 64 samples should be adequate.

Set up the DMA multiplexer to route the DMA request from the ADC to the selected DMA channel. Configure the source address for the channel to be the ADC data register with no address update after each transfer. The destination address should be the beginning of the buffer with address increment and modulo update after each transfer.

The DMA controller supports independently configurable source and destination data sizes or 1, 2 or 4 bytes. Although the ADC data register presents 16-bit data, it may be read as a 32-bit word with the MS 16 bits set to 0. Read ahead and understand the requirements of the rest of the project before making a configuration decision here.

The DMA controller can be programmed to stopped after a specified number of transfers. Since continuous operation is desired, disable this feature by clearing the D-REQ bit in the DMA control register (DCR).

With DMA configured and enabled and the ADC running, the buffer will be continuously updated with data from the ADC. Use a memory contents report like you wrote for Project 1 to interactively monitor the contents of the buffer as you apply a signal to the ADC input.

## Questions

1. Can you measure any change in CPU processing speed when DMA is running compared to when it is not? How did you perform this comparison?
2. What source and destination DMA read and write size did you choose and why?

# Part 4: Double buffer

A double buffer is a pair of buffers, often adjacent in memory. Like the circular buffer, double buffers are used for buffering data and synchronization between two tasks. At any one time, one buffer is owned by one task and the other is owned by the other task. Once both tasks have finished their processing of the data in their buffer, the buffers are swapped and the next processing operation begins.

In this project, one of the buffers will be owned by the DMA process and the other by your application or interrupt service routine which will process the data.

The modification required to move from the single-buffered system implemented in Part 3 to the double-buffered implementation is to divide the buffer in half and establish some signaling to swap buffers. The signal in this case takes the form of an interrupt generated by the DMA controller based on transfer count.

Configure the DMA controller to generate an interrupt when half the buffer is filled. Acknowledge the interrupt and reconfigure to generate another interrupt when the other half has been filled. Repeat. Toggle a GPIO pin in the ISR and monitor with an oscilloscope to verify expected frequency and consistency of the interrupt service. **Take a screenshot of your oscilloscope output and include it in your lab report.** Write test code in your ISR to read the DMA write pointer and verify it is in the expected location when the interrupt occurs.

# Part 5: Application

Write an application examines each sample in the input data and performs analysis on the data. Two real-time results are produced: peak level and RMS level.

Peak level is a measurement the absolute value of the highest sample value observed over an interval. To produce a continuously updated peak level reading, the reading is increased instantaneously when a sample with an absolute value higher than the current peak level reading is encountered. Otherwise, peak level is decreased using a first-order decay to a zero.



Where the coefficient alpha is between 0 (instant decay) and 1 (infinite hold). The decay calculation can be performed once per buffer period.

RMS level is the square root of the sum of the squares of the sample values divided by the number of samples.

.

To produce a continuously updated RMS level a first-order low-pass filter (exponential moving average) can be used in place of the mean operation.



Working with squares of sample values requires using twice as many bits than is present in the input data. This is a manageable requirement since samples from the ADC are 16 bits and processor is arithmetic is 32 bits.

The square root operation required to get a final results presents a potential computational challenge. Solutions are discussed in our text and lectures. You may dodge this calculation by redefining our measurements as [power rather than level](https://en.wikipedia.org/wiki/Field,_power,_and_root-power_quantities).

The application should update these readings for each buffer of samples processed. The application should provide continuous updates of these two readings as text through the UART as frequently as the output baud rate allows.

## Questions

1. What effect does changing the buffer size have on system processing efficiency?
2. What other effect(s) does changing the buffer size have on the system?

## Extra credit

1. Perform a square root calculation using Taylor series to report true RMS value. (Also report the square version so graders can verify your results.) (10 points)
2. Perform a logarithmic calculation using a lookup table and linear interpolation to report signal levels in [dBFS units](https://en.wikipedia.org/wiki/DBFS). (Also report the linear version so that graders can verify your results.) (10 points)

# What to turn in

1. Brief lab report with block diagram and oscilloscope screenshot
2. Code dump report in txt or PDF format
3. Answers to questions posed here in the project description
4. Github repo link, should be tagged
5. KDS zipped project

# Grading rubric

1. Block Diagram: 10 points
2. Oscilloscope screenshot 10 points
3. Final application: 30 points
4. Lab questions: 20 points
5. Demo: 30 points
6. Extra credits(max): 20 points