# PES Project 6 – FreeRTOS, ADC, DAC, DMA, DSP - Total 100 Points (plus 10 bonus points)

## Introduction

In this final PES project, you will generate a sine wave signal on the KL25Z's DAC, capture the signal with the KL25Z's ADC, transfer collected data buffers via DMA to another data buffer, and perform basic DSP analysis of the signal. You will exercise lookup tables, buffers, ADC, DAC, DMA, math functions, and basic FreeRTOS task management. You will optionally use an oscilloscope or logic analyzer to verify output. For extra credit, you will employ FreeRTOS mutex, semaphore, and queue functions.

This project has two deliverable programs, one for the confirmation of the creation of sine wave data for the DAC and the second for full functionality for reading, transferring, and analysis of that data.

The programs should be able to be demonstrated in debug and normal modes. In debug mode, comments on program steps should be provided along with any values needed to verify proper operations. In normal mode, much less verbose logging should be used.

## General

As with most of the PES projects, there are details for this project that are left to you to decide, for instance, the buffer structure for the ADC data capture. Document the design decisions you make to meet the requirements of the project in the Observations section of the project README.

You can reuse your buffer code from Project 5. You can also reuse your logger code from Project 5 (switching to use PRINTF rather than the custom UART for output to the console). You will update logger enums for the new code structure of this project, and you must show timestamps based on time since program execution started. You may use any LED control code (bare metal or SDK based). In the event of ANY error condition in any part of the project, set the LED to steady red.

For this project you may use all available SDK and FreeRTOS functions. You should use the SDK supplied FreeRTOS libraries and include files.

Your code should follow the ESE C Style Guide as closely as possible.

When compiling use -Wall and -Werror compiler flags. Your code should have no compilation errors or warnings.

You should use modular design for your code – drivers in one module, application code in another, etc.

## Program 1 – Lookup Table Creation and Validation

Calculate and create a lookup table to represent the values in a sine wave that runs from 1V to 3V. Period from peak to peak should be 5 seconds with a .1 second step. In code, mathematically determine the voltage values at each .1 second step. Convert the voltages to DAC register values representing the voltages at each .1 second step. Store the DAC register values in a buffer of sufficient size to hold all the values for a single peak to peak cycle.

Create a test case that uses a FreeRTOS Software Timer Callback function to apply the values from the lookup table to DAC0_OUT (pin J10-11) every .1 second, repeating from the beginning of the table once the last value is applied. Toggle a Blue LED on and off for each visit to the timer callback.

Confirm the shape of the output by looking at printed values in a console window or by charting the data on a scope; voltages should be 1V to 3V, peak to peak period should be 5 seconds, signal should be a clear sine wave. You may optionally capture a scope image for submission (label it Program 1).

When run in debug, program steps and values should be echoed to the console by logger statements.

## Program 2 – Capture the output from DAC0 (pin J10-11) on ADC0 (pin J10-1, ADC0_SE8)

Initialize this program by using the code from Program 1 to create a lookup table for use by the DAC FreeRTOS task, putting the register values that define the sine wave into a DAC value buffer of appropriate size.

Create and start the following FreeRTOS tasks. Note that you will need to determine appropriate task priorities and scheduling, as well as any need for mutex or semaphore use for communication or resource sharing, in order for the program to meet its performance goals.

Create a FreeRTOS Task to periodically (every .1 seconds) change the value on DAC0 from the lookup table buffer of DAC register values. Toggle a Green LED on and off for each DAC write (if the LED is available for use).

Create a FreeRTOS Task to periodically (every .1 seconds) read the DAC0 value via ADC0 and store it in a circular buffer. The ADC buffer will be 64 samples long and should contain the raw ADC register values from each read. When the buffer is full initiate a DMA transfer from the ADC buffer to a second buffer (called the DSP buffer). When the DMA Transfer is about to start, toggle the LED to Blue for .5 seconds. During this period, the LED cannot be used by other tasks. Clear (or overwrite) the ADC buffer with incoming DAC values and continue sampling until the next series of samples are collected. Capture a time stamp at the start and completion of the DMA transfer. You will need to consider the size and data width requirements for the ADC buffer and the DSP buffer.

Create a FreeRTOS Task that is triggered by an interrupt from completing the DMA transfer of data into the DSP buffer. Calculate the following floating point values from the ADC register values: maximum, minimum, average, and standard deviation of voltage levels. You may see slight (approx. 2%) error in the ADC reading the incoming DAC values. Report those values along with an incremented run number starting at 1 and the start time and end time for the last DMA transfer. Once run number 5 is completed and reported, terminate the DAC and ADC tasks, and terminate this task to end the program.

Optionally confirm the shape of the ADC values on a scope; voltages should be very nearly 1V to 3V, peak to peak period should be 5 seconds, signal should be a clear sine wave. Optionally capture a scope image for submission (label it Program 2).

When run in debug, program steps and values should be echoed to the console by logger statements.

## Extra Credit

There is a 10 point extra credit opportunity: 5 points to use a FreeRTOS Mutex or Semaphore to control access to the shared LED and 5 points to use a FreeRTOS Queue in place of one of the use of your custom buffers.

## Code References and Approach

You'll want to review the RTOS examples provided with the SDK for FreeRTOS timer, synchronization, and task management. (See lecture notes for updating the SDK for FreeRTOS use.) You'll also want to look at the samples under demo_apps called dac_adc for setup of the ADC and DAC subsystems and under driver_examples, you'll find a sample for memory to memory transfers using DMA. A key to this project is confirming each individual part of the program before building the entire application in FreeRTOS.

## Project Submission

The project is due on **Tuesday 4/28** prior to class and will be submitted on Canvas. The project will also be demonstrated to class staff in recorded videos or web conferences. We will be setting up demo slots to allow a detailed review of the submissions with the student teams. The Canvas submission will consist of two parts:

Part 1 is a single GitHub repo URL which will be accessed by graders to review code and documentation. This will consist of any C code or include files, captured output files, and a README Markdown document that includes:

- A title (PES Project 6 Readme)
- Names of your team
- A description of the repo contents
- Observations: A description of any issues or difficulties you encountered on the project and how they were addressed, as well as any design decisions you made to meet project requirements
- Installation/execution notes: for others who may use the code – this should include compilation instructions for the SAs to more easily grade
- The repo should contain the two scope images mentioned above
- Please include a Git tag called Final on your final submission to allow the SAs to be clear about what was submitted for grading
- Please note – code being pushed to Git should not be zipped – we should be able to pull project folders directly from your Git repos.

Part 2 will be a PDF containing all C code and README documentation – the PDF is used specifically for plagiarism checks: your code should be your team's alone, developed by your team. You should provide a URL for any significant code taken from a third party source, and you should not take code artifacts from other student teams. However, you may consult with other teams, the SAs, and the instructor in reviewing concepts and crafting solutions to problems (and may wish to credit them in documentation).

## Grading

Points will be awarded as follows:

- 40 for the correctness of demonstrated code (execution of the two programs in debug and normal mode) **– code will be demonstrated remotely with SAs on or after the project due date**
- 50 for the construction of the code (including following style guide elements and the quality of solution)
- 10 points for the README
- 10 points possible extra credit for use of FreeRTOS queue and mutex/semaphore

Project 6 is due on Tuesday 4/28 at 2 PM. Assignments will only be accepted late for two days until Thursday 4/30, the final day of classes, at a penalty of 5% of the grade (after the four hour grace period). After that point, assignments will not be accepted. Please note, due to the close of the semester and the need to complete grading, Project 6 has a smaller than normal late window and there will be NO EXTENSIONS to the Project 6 due date. Please plan accordingly.