ECGR 4101/5101, Fall 2016: Lab 5

Creating Your Own Library and Analog to Digital Converters

Version 2 – 2016-10-23 (change only in the lab checkoff page – see red text)

Objective:

In this lab, you will learn how to create your own C library files, and call an ADC reading from a function contained within it. The initialization and analog reading code will be created by setting individual registers without the use of any driver functions provided by Renesas.

Assignment:

1. Begin by creating an empty project workspace. Consult the supplemental instruction from the previous lab if you need help doing this. If you do, at the step where you select "Tutorial Project", select "Empty Application" instead and click "Finish."
2. Once the project is set up, go to file->new to open a blank file. We will save this file as "customADC.h" under the src directory in your workspace. We will then open another blank document and save it in the same directory as "customADC.c". We do this because libraries will usually have a .h file to accompany a .c file.
3. Now we will populate the .h files with the function prototypes. Header files contain #define macros, function prototypes, and sometimes, but not often, entire functions. Under normal circumstances, functions should only be written within the .c file which the header file corresponds to.
4. In your header file, include the board's header file
    a. #include "platform.h"
5. In your header file, create 2 function prototypes:
    a. void adcInit(void); // This function initializes the ADC for us
    b. int adcRead(int channel); // This function will read an analog value from the specified channel

    adcInit() will be used to initialize the 12 bit ADC by starting the peripheral clock, selecting alignment settings, mode, etc. adcRead() will be used to initiate an ADC conversion on the channel specified by the "int channel" parameter, wait for the conversion to finish, then return the result.
6. Now in the customADC.c file, we will implement these functions.
    a. In adcInit() you will need to do the following:
        i. Power on the 12 bit ADC peripheral by enabling it's clock
        ii. Set up the I/O pins the ADC inputs are attached to
        iii. Configure the 12 bit ADC Control Register for single cycle mode, no interrupts
        iv. Configure the data alignment register
        v. Configure the ADC trigger to use software triggering
    b. In adcRead() you will need to do the following:
        i. Select the 12 bit ADC channel to read from
        ii. Begin conversion

iii. Wait for conversion to finish
iv. Return result

To learn about the 12-bit ADC register, be sure to consult the data sheet, the course textbook, and even the tutorial code for information on how to create these functions.

7. Once you have finished creating the functions, add the adcInit() function somewhere in your main() before the while(1). Add adcRead() in the while loop and pass it 0x0004 to read from the channel that the potentiometer is attached to. Take the ADC value returned by the adcRead() function, convert it to a string, and write it to the display using the lcd_display() function. To do this, you will need to learn how to use the sprintf() function.
    a. Note: sprintf() will compile without including the stdio.h library which it needs. If you are getting unexpected text printed to your LCD, be sure to check that this library is included.
    b. Also Note: The lcd_display () function can accept a pointer to a string as the second argument. Use sprintf() to create the text which includes variable values you want to display on the LCD.
8. Now you can test your custom ADC functions. Be sure everything is working before continuing on from this point.
9. We will do a little more abstraction before moving on from this point. In customADC.h, let's add some defined macros to pass to the adcRead() function for channel selection. Make definitions for AN0, AN1, … AN15. That way we can call adcRead(AN2); to read from channel 2. Creating definitions removes "magic numbers" from your code and makes it easier for yourself and other programmers to read.
10. Let's add another function to our .c and .h files to convert the digital value to the voltage value. Create a convertADC() function. This function will be passed the ADC value as an int, and return the voltage as a float. Make the function flexible, so that if in the future we can use this function for converting ADC values from the 10 bit ADC as well, along with various reference voltages. We want to create robust functions so that we will be able to use them for various purposes in the future. You will be graded on how well you design your function.
11. For the final demonstration, your program will need to have the following requirements:
    a. Display your team's name on the first line of the LCD
    b. On the second line, display "ADC Test"
    c. When switch 1 is pressed, read the voltage from channel 2 (the potentiometer's channel) and have the third line display "Channel 2:" and the fourth line display the converted ADC value.
    d. When switch 2 is pressed, take a reading from an ADC channel of your choice and display the channel number on the third line and the ADC reading from that channel on the fourth line.
    e. When switch 3 is pressed, take a reading from a different ADC channel of your choice and display the channel number on the third line and compare the reading from the switch 2 press channel and display 'HIGH' if it's greater or 'LOW' if the value is lower or 'EQUAL' if the ADC values are equal.

Make sure you select an ADC channel that has a pin available on one of the board's headers so we can connect a voltage supply to test.

To find out what ADC pins are available, check the hardware manual section 22 AND check the schematic of the YRDK board.

To Submit:

- Create a text file containing all the functions and upload it to Canvas
- Your lab check-off sheet at the demonstration

Grading will be changed such that 50% is your demonstration, and the remaining 50% will be your cod structure. Your code will be graded based on comments, proper alignment (indentations), function design, proper capitalization of definitions and variables, and design of custom functions.

Embedded Systems Lab Demonstration Validation Sheet

This sheet should be modified by the student to reflect the current lab assignment being demonstrated

Lab Number:	Lab 5 – ADC and H Files
Team Members	Team Member 1: Team Member 2:
Date:	

Lab Demonstration Requirements

REQ Number	Objective	Self-Review	TA Review
1	Team name and ADC Test is displayed		
2	SW1 reads the correct ADC value from channel 2		
3	SW2 reads the correct ADC value from a user defined channel		
4	SW3 reads from a different user channel , compares the voltage from the first user channel and displays HIGH, LOW, or EQUAL		

Code Requirements (will not be graded during lab demo)

REQ Number	Objective	Self-Review	TA Review
1	customADC.h and customADC.c should be constructed as specified in the handout. All functions must follow the specified argument and return types and include all functions specified. You may write additional functions for your library, but they must follow standard C coding practices, including capitalization, indentation, and commenting.		
2	adcInit()must be written as specified by the lab handout and follow the given steps.		
3	adcRead() must be written as specified by the lab handout and follow the given steps.		
4	convertADC()must be written as specified by the lab handout.		
5	All code must be commented and indented properly.		