ECE 315: Computer Interfacing

Lab #3: Introduction to Analog-Digital Converters, Digital-Analog Converters and Photocell Light Sensors

Conducted on: February 27, 2020

Conducted by: Kathleen Pescador, Abuni Gaiya

Lab Section: H41

Table of Contents

Abstract

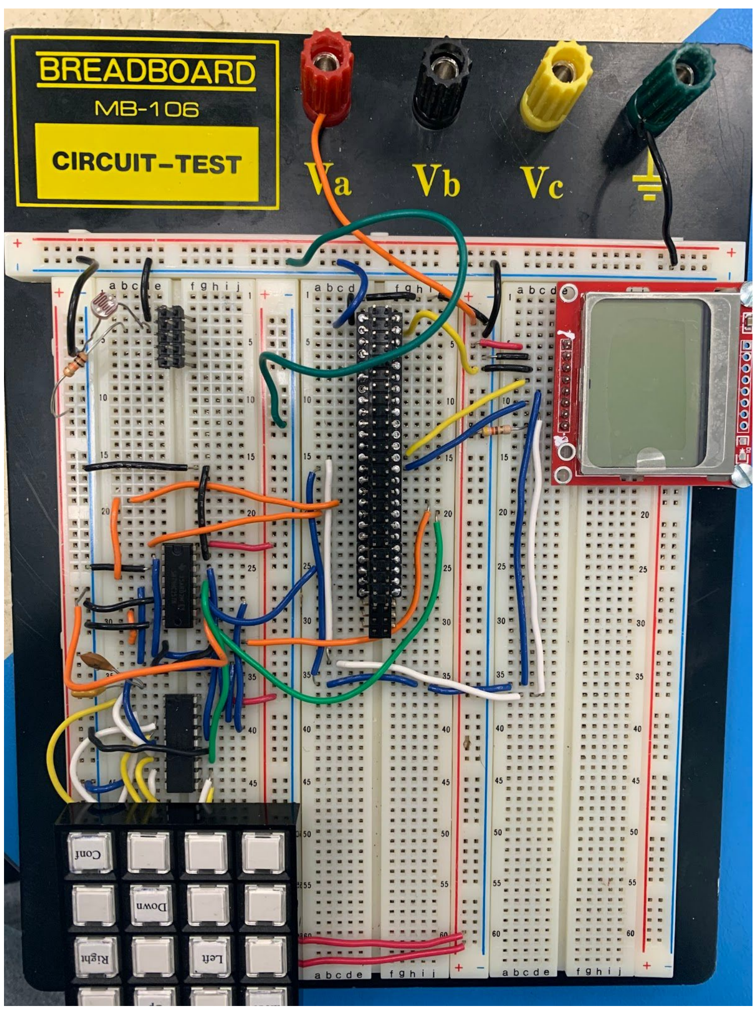
For this lab, our goal was to get familiar with the analog-digital and digital-analog converters on the MCF54415-based Netburner boards. We were also to gain experience with a simple sensor that detects light and to demonstrate context switching using specially crafted tasks. This lab involves a light meter application, where we would use a sensor to create a simple light meter. This lab consists of four parts. We would have to wire our LCD according to the schematic and suggested floor plan that are provided in eClass. In the first part of the lab, we were to build a new project and verify that our circuit was wired correctly. We could do this by using the lab3 files from eClass to display a welcome message. For the second part of the lab, we would have to complete the AD class from the provided code at the recommended locations to read the data from the photocell. This part would enable us to see the light level readings of our photocell from the serial console. On the other hand, the third part of the lab requires us to create the LCD code that displays the input from the photocell in a bar-graph format to the LCD. Whereas the last part of this lab would need us to create the DA code that outputs three different voltage levels corresponding to the three separate tasks in the provided code. We would have to measure and record the time taken for each of the three tasks, and the order they executed in. We then have to modify the tasks and retake the measurements.

Design

Exercise 1

Exercise 1 was basically testing the circuit which we had designed in the prelab, as shown in *Figure 1* and making sure we had designed it properly. For this lab we added three extra components, a 16-33K Ohm Photocell, a 10K Ohm Resistor and a 10-pin header to the circuit we had been building from the previous labs. We connected the 50 -pin cable and 10-pin cable from the ColdFire microprocessor to their respective headers and ran the lab3 code we downloaded from eclass. The project was run and then we checked the heartbeat and context switching pins (J2[3] and J2[4]). We were also checking to make sure that the default web page served. As we measured a voltage of 3.3V and LCD displayed the text “welcome” once the project was run.

Figure 1: Hardware Circuit Setup



Exercise 2

The second part of the lab involves completing the AD class from the provided code obtained from eClass. But first, we had to test our photocell circuit and ensure that the voltage varies between 0 and 3.3 volts maximum depending on light conditions. We placed our hand over the photocell and checked that the voltage varied within that range. Once we have that verified, we completed the following AD methods from AD.cpp. This is shown as follows:

The **AD::StartAD** method takes in no input and output, and also starts the AD conversion and only a single sample is converted:

```
void AD::StartAD(void){
     sim2.adc.sr = 0x0000;
     sim2.adc.cr1 = ADC_CR1_START|ADC_CR1_LOOP; }
```

The **AD::StopAD** method takes in no input and output and is responsible for turning off the AD converter:

```
void AD::StopAD(void) {
     sim2.adc.cr1 = ADC_CR1_STOP; }
```

The **AD::ADDone** method takes in no input while checking the status of the AD conversion. Thus, this returns a boolean value which returns true if the AD conversion is done and false if not done.

```
bool AD::ADDone(void) {
     return sim2.adc.sr && 1; }
```

The **AD::GetADResult** method takes in no input and output, and also reads the result of the AD conversion once the status register indicates that the conversion is done:

```
WORD AD::GetADResult(int ch) {
     return sim2.adc.rslt[ch]>>3; }
```

After completing the AD methods, we completed the UserMain *while* loop that would implement the pseudocode from the lab manual. We then displayed the values read in from AD to our serial console. We modified the light levels over the photocell by cupping a hand over the sensor or using a smartphone flash close to the sensor. This would vary the light that will be detected, and thus modifying the results read from the serial console.

The following is our implementation of the code inside the UserMain method in lab3.cpp:

```
while (1) {
      WORD result = 0;
        float volts = 0.0;
        myDA.Lock();
      myDA.DACOutput(USER_MAIN_VOLTS);

      myAD.StartAD(); //StartAD
      while (!myAD.ADDone()){
            OSTimeDly(TICKS_PER_SECOND/20);
      }  //Busy Wait until done
              result = myAD.GetADResult(0);//Get Result
              myAD.StopAD();//Stop the AD
              iprintf("RESULT %d\n", result);
              int my_result = result;
              int brightness[] = {50, 271,492, 713,934,
1155,1376,1597,1818,2039,2260,2481,2702};
              myLCD.Clear();
              for (int i = 0; i<12; i++){
                    if (my_result <= brightness[i]) {
                          myLCD.DrawBarGraph(2,i);
                          break; }
              }
      myDA.DACOutput(0);
      myDA.Unlock();
        OSTimeDly(TICKS_PER_SECOND/20); }
```

Exercise 3

Exercise 3 could be viewed as the main part of this lab, here we created the LCD code that displays the input from the photocell in bar-graph format to the LCD. After being able to display the values read from the photocell from the different light input variations we noticed that our values never went beyond 48 (for the brightest light ie when we flashed a light pointed to the photocell) and 2707 (for the dimmest light ie when your hand is cupped around the photocell). Using these values we set our boundaries for our photocell to be 50 for the dimmest light and 2702 for the brightest light.

We implemented the LCD::DrawBarGraph method and modified the UserMain to complete this part. We started off by implementing the LCD::DrawBarGraph method and testing that it works with by passing hardcoded values to the method in lab3.cpp. For this function, we defined our usual sprite which we have been using since lab 1, then we also defined an array called start which stored the start point for each of the 6 lines of the LCD. Given a line and a length for the bar graph we would draw the given sprite on the line for the specified length as shown in the code block below:

```
void LCD::DrawBarGraph(BYTE line, BYTE length){
    BYTE sprite[] = {0xAA, 0x55,0xAA, 0x55,0xAA, 0x55,0xAA};
    BYTE start[] = {0, 12,24, 36,48, 60};
    while(length>0){
     DrawChar(sprite, char_index[start[line]]);
     DrawChar(sprite, char_index[start[line] + length]);
     Length--; }
}
```

When we were sure that our LCD::DrawBarGraph method was working properly we modified the userMain to map the value of the result gotten from the analog to digital converter based on the mapping table shown in [Appendix I: Various Brightness Levels for Photocell and their corresponding ranges](#) and draw the corresponding bar graph on line 2 of the LCD. Our implementation for this part is shown in the code block below:

```
 int my_result = result;
 int brightness[] = {50, 271,492, 713,934,
1155,1376,1597,1818,2039,2260,2481,2702};
             myLCD.Clear();
             for (int i = 0; i<12; i++){
                 if (my_result <= brightness[i]) {
                     myLCD.DrawBarGraph(2,i);
                     break; } }
```

Exercise 4

The last part of the lab involves creating a DA code that outputs three different voltage levels corresponding to the three separate tasks in the code package. We started off by turning the optimization level in our project settings to none. This would allow the given task code to execute without getting opted out.

We then completed the **DA::DACOutput** method from DA.cpp which would send the data out to the DA converter. We converted the volts from a float type to the 12-bit integer value that the DA expects. Our implementation of the code is shown below:

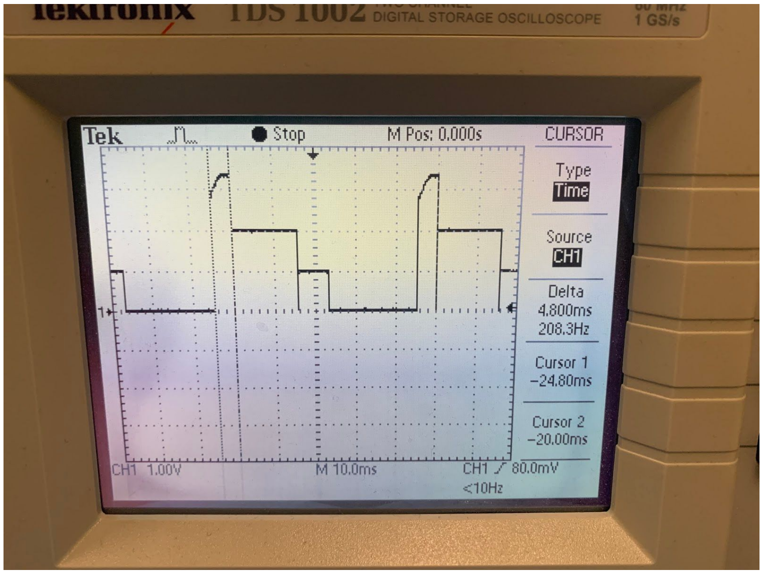
```
void DA::DACOutput(float volts) {
      unsigned int X = (volts/3.3)*4096;
      sim2.dac[0].data = X ; }
```

After that, we had to modify all delays within our code to be able see all distinct values on the oscilloscope.

```
OSTimeDly(TICKS_PER_SECOND/20);
```

We connected the DAC0 pin to the oscilloscope and obtained a plot as shown in *Figure 2*. All plot readings are shown in Appendix III: Oscilloscope Readings taken from Exercise 4.

Figure 2: Oscilloscope Reading Sample from Exercise 4



We measured the time taken for normal operation of all three tasks and also recorded the order in which these tasks are executed in. After doing so, we modified the *for* loop constants in task 1 and task 2 by doubling each of them and once again did the time measurements and recorded their order. The results read from the oscilloscope are shown in Table 1.

Table 1: Raw data from recording the time of each task shown in Figure 2

	UserMain time	Task 1 time	Task 2 time
Normal Loop Constant	4.80 ms	16.00 ms	7.60 ms
Doubled Loop Constant	4.80 ms	30.40 ms	15.60 ms

Testing

Case	Description	Expected Results	Experimental Result	Conclusion
Exercise 1: To build a new project for lab 3.	i. Connect the 50-pin cable and 10-pin cable from the ColdFire microprocessor to your breadboard. Run the default lab 3 code. iii. Check the heartbeat and context switching pins (J2[3]andJ2 [4]). Iv. Verify that the default webpage is being served.	The default webpage is being served and the voltage is 3.3V.	The default webpage is being served which displays the text 'welcome' and the voltage is 3.3V.	Test passed
Exercise 2: Complete the AD class at the recommended locations to read the data from the photocell.	i.Complete the AD methods from AD.cpp ii. Test by cupping your hand around photocell and flashing torchlight at photocell	i. The serial console displays a value less than or equal to 50 when a bright light is flashed ii. The serial console displays a value around 2700 when hand is cupped around photocell	i. The serial console displays a value of 47 when a bright light is flashed ii. The serial console displays a value of 2702 when hand is cupped around photocell	Test Passed
Exercise 3: Create the LCD code that displays the input from the photocell in bar-graph format to the LCD	i. Modify the LCD class to complete the DrawBarGraph method ii. Complete userMain to display level of brightness/dimness on screen iii. Test by cupping your hand around photocell and flashing torchlight at photocell	i. LCD displays no bar when bright light is flashed. ii. LCD displays midway bar when nothing is done. iii. LCD displays a full bar when the hand is cupped around the photocell.	i. LCD displays no bar when bright light is flashed. ii. LCD displays midway bar when nothing is done. iii. LCD displays a full bar when the hand is cupped around the photocell.	Test Passed as shown in Appendix II: Images showing the Three Test Cases for Exercise 3.
Exercise 4: Create the DA code that outputs three different	i. Complete the DA code from DA.cpp ii. Connect to the Oscilloscope and take the	i. The oscilloscope shows a figure with three different peaks due to the different voltages	i. The oscilloscope shows a figure with three different peaks. The order of this code is what	Test Passed as shown in Appendix III: Oscilloscope Readings

voltage levels.	time measurements and order for each task. Do this for both the normal loop constand and the doubled loop constant.	for each task. The order of this code should be happening at 3.0V (UserMain), 2.0V (Task1), and 1.0V (Task2). Because the loop of Task 2 is about half of Task 1, it should have about half of its time as well. ii. When the loop constant is doubled, the order of the tasks are to be unchanged, and the time measured for Task 1 and 2 should be doubled.	was expected, which goes from 3.0V (UserMain), 2.0V (Task1), and 1.0V (Task2). The time measure for Task 2 is about half as the time for Task 1. ii. When the loop constant is doubled, the order of the tasks are unchanged, and the time measured for Task 1 and 2 are doubled. The UserMain time is unchanged.	taken from Exercise 4

Conclusion

In conclusion, this lab was a very interesting lab for us We learned a lot form about using the LCD display, MCF54415 microcontroller, the oscilloscope, and all the other tools and equipment we have been using in the lab from lab 1 and lab 2 and at the end of lab 3 we feel really confident using them. We were successfully able to design a light sensing device that shows how much light there is on an LCD screen. We were also able to see a practical example of how analog-digital converters and digital-analog converters work. And finally see a demonstration of context switching using specially crafted tasks. At the end of this lab we tested all the requirements for the lab and all our tests were successful. Overall, this lab was a success.

Appendix:

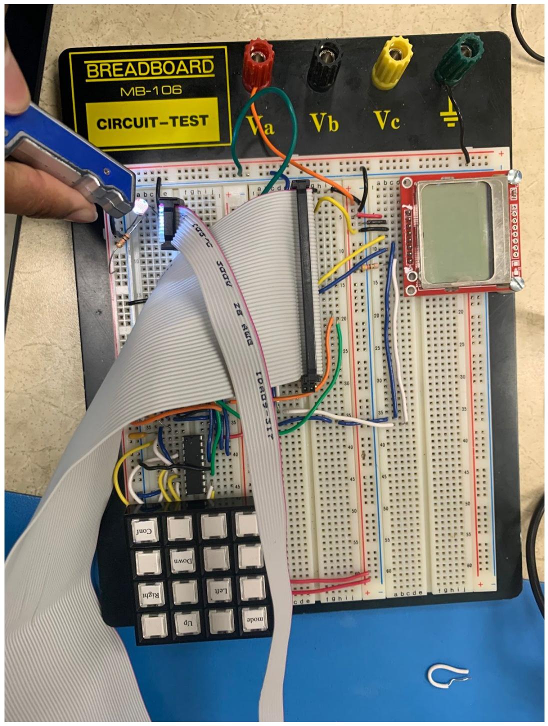
I. Various Brightness Levels for Photocell and their corresponding ranges.

Table 3: Table Showing Various Brightness for Photocell and their corresponding ranges.

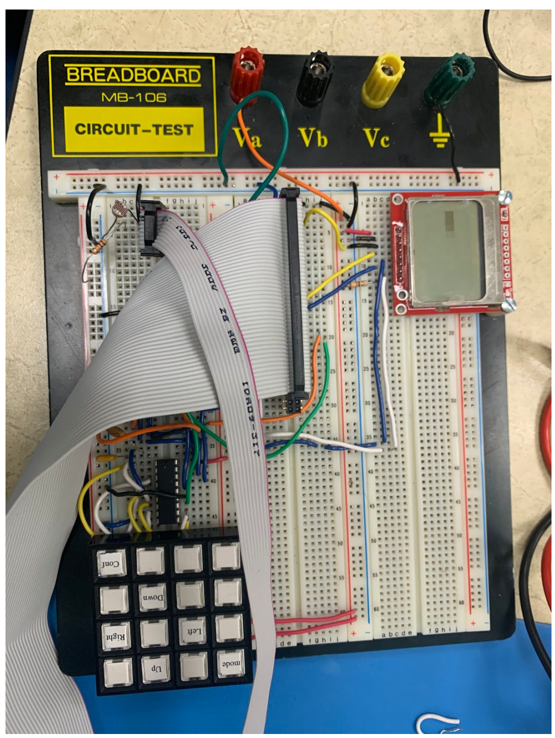
Number from ADC less than ______	**Brightness Level**	**Description**
50	0	*Brightest case when flashlight is directly on top of photocell*
271	1	*Second Brightest*
492	2	*Third Brightest*
713	3	*Fourth Brightest*
1155	4	*Fifth Brightest*
1376	5	*Sixth Brightest*
1597	6	*Seventh Brightest*
1818	7	*Eighth Brightest*
2039	8	*Ninth Brightest*
2260	9	*Tenth Brightest*
2481	10	*Eleventh Brightest*
2702	11	*Dimmest case when hand is cupped around photocell*

II. Images showing the Three Test Cases for Exercise 3.

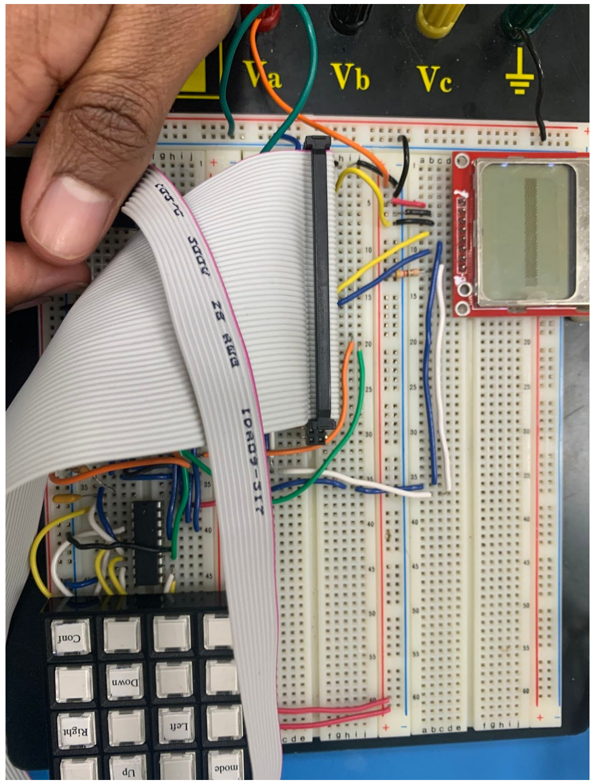
Test Case 1: Extremely Bright Light



Test Case 2: Moderate (Natural) Light



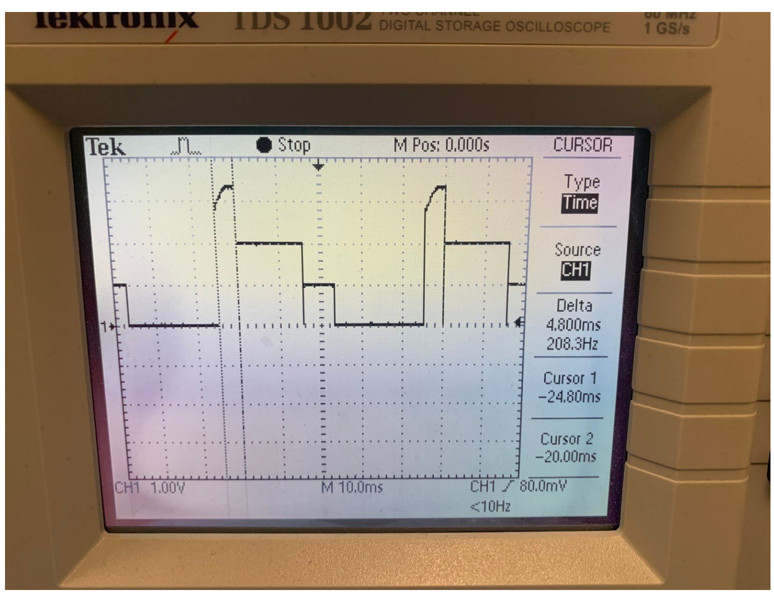
Test Case 3: No (Dim) Light



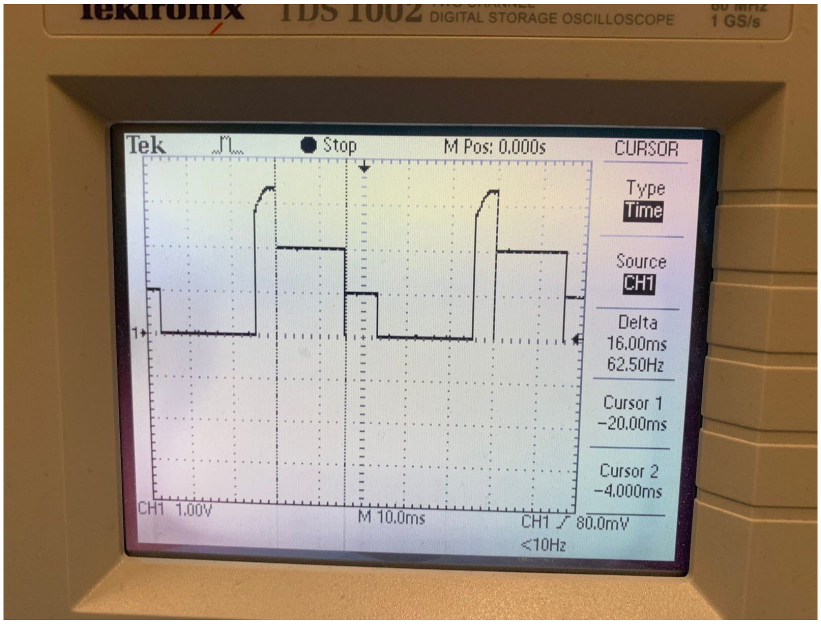
III. Oscilloscope Readings taken from Exercise 4

A. Normal Loop

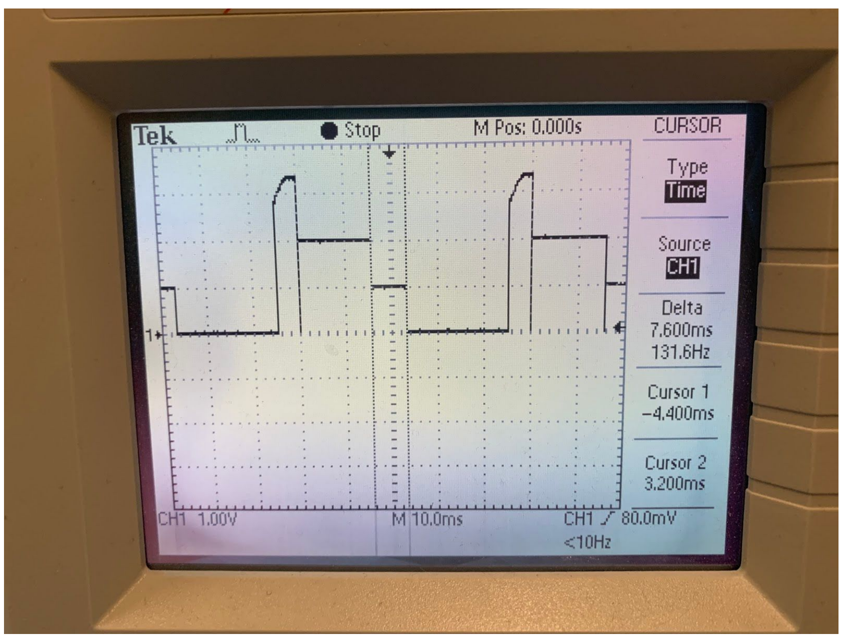
1. UserMain



2. Task 1

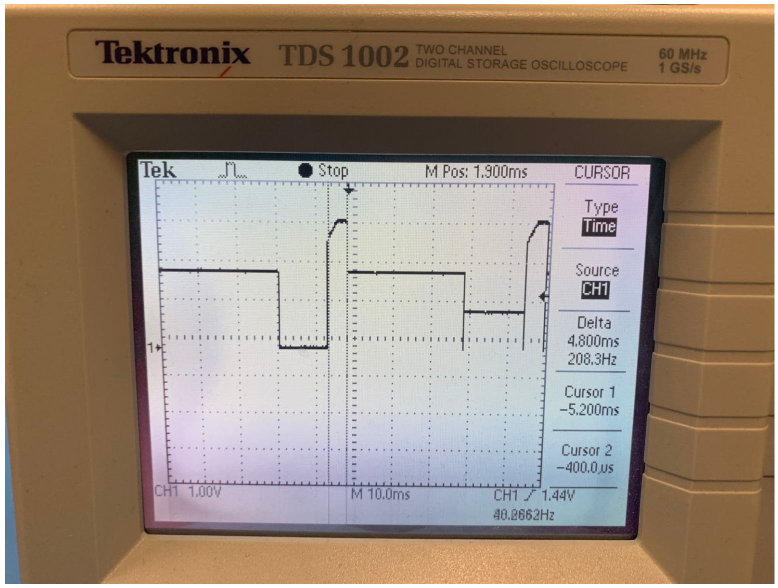


3. Task 2

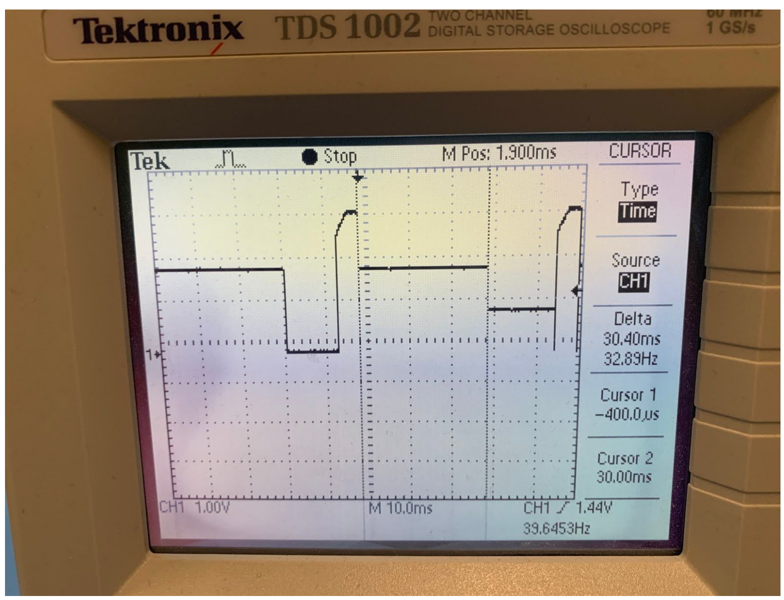


B. Double Loop

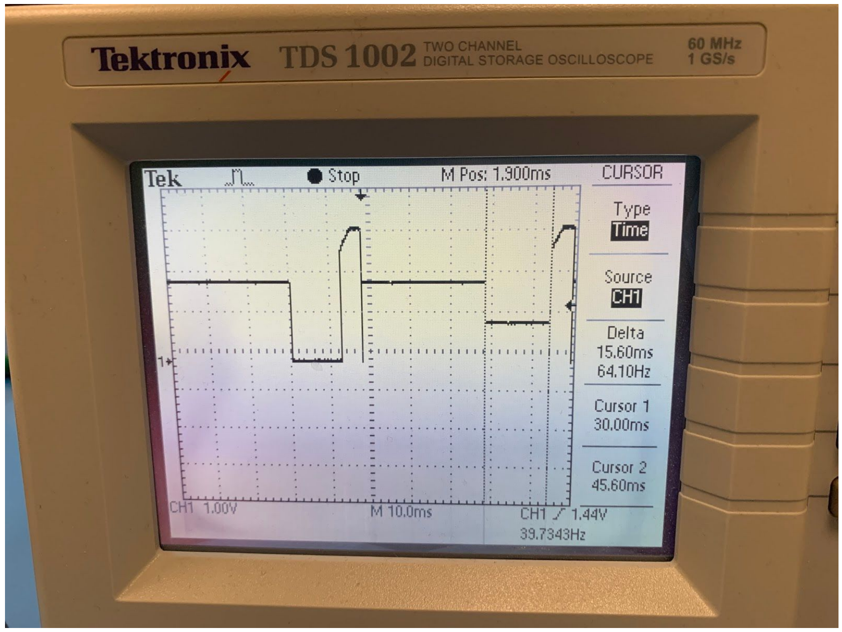
1. UserMain



2. Task 1



3. Task 2



IV. Modified Code

Lab3.cpp

```
#include <predef.h>
#include <stdio.h>
#include <ctype.h>
#include <startnet.h>
#include <autoupdate.h>
#include <smarttrap.h>
#include <taskmon.h>
#include <NetworkDebug.h>
#include "AD.h"
#include "DA.h"
#include "LCD.h"

extern "C" {
void UserMain(void * pd);
void StartTask1(void);
void Task1Main( void * pd);
void StartTask2(void);
void Task2Main( void * pd);
```

```
}

#define USER_MAIN_VOLTS 3.0
#define TASK1_VOLTS     2.0
#define TASK2_VOLTS     1.0

/* User task priorities always based on MAIN_PRIO */
/* The priorities between MAIN_PRIO and the IDLE_PRIO are available */
#define DUMMYTASK1_PRIO     MAIN_PRIO + 1
#define DUMMYTASK2_PRIO     MAIN_PRIO + 2

/* Task stacks for all the user tasks */
/* If you add a new task you'll need to add a new stack for that task */
DWORD DummyTask1Stk[USER_TASK_STK_SIZE] __attribute__( ( aligned( 4 ) ) );
DWORD DummyTask2Stk[USER_TASK_STK_SIZE] __attribute__( ( aligned( 4 ) ) );

const char * AppName="Abuni AND Kathleen";

AD myAD;
DA myDA;
LCD myLCD;

void UserMain(void * pd) {
    InitializeStack();
    OSChangePrio(MAIN_PRIO);
    EnableAutoUpdate();
    StartHTTP();
    EnableTaskMonitor();

    #ifndef _DEBUG
    EnableSmartTraps();
    #endif

    #ifdef _DEBUG
    InitializeNetworkGDB_and_Wait();
    #endif

    iprintf("Application started\n");
    myLCD.Init();
    myLCD.Home();
    myLCD.Clear();
    myLCD.DrawString("Welcome");
    myAD.Init();
    myDA.Init();
```

```
    myDA.DACOutput(0);
    StartTask1();
    StartTask2();


    while (1) {
        WORD result = 0;
        float volts = 0.0;
        myDA.Lock();
        myDA.DACOutput(USER_MAIN_VOLTS);

        myAD.StartAD(); //StartAD
        while (!myAD.ADDone()){
            OSTimeDly(TICKS_PER_SECOND/20);
        }
       //Busy Wait until done
             result = myAD.GetADResult(0);//Get Result
             myAD.StopAD();//Stop the AD
             iprintf("RESULT %d\n", result);
             int my_result = result;
             int brightness[] = {50, 271,492, 713,934,
1155,1376,1597,1818,2039,2260,2481,2702};
             myLCD.Clear();
             for (int i = 0; i<12; i++){
                 if (my_result <= brightness[i]) {
                     myLCD.DrawBarGraph(2,i);
                     break;
                 }
             }








        myDA.DACOutput(0);
        myDA.Unlock();
        OSTimeDly(TICKS_PER_SECOND/20);
    }
}

/* Name: StartTask1
 * Description: Creates the task main loop.
```

```
 * Inputs: none
 * Outputs: none
 */
void StartTask1(void) {
    BYTE err = OS_NO_ERR;

    err = display_error( "StartTask1 fail:",
                    OSTaskCreatewName(  Task1Main,
                    (void *)NULL,
                    (void *) &DummyTask1Stk[USER_TASK_STK_SIZE],
                    (void *) &DummyTask1Stk[0],
                    DUMMYTASK1_PRIO, "Dummy Task 1" ));
}

/* Name: Task1Main
 * Description: NOP loop that can be timed
 *  Does nothing useful
 * Inputs:  void * pd -- pointer to generic data . Currently unused.
 * Outputs: none
 */
void    Task1Main( void * pd) {


    DWORD count = 0 ;


    /* place semaphore usage code inside the loop */
    while (1) {

        myDA.Lock();
        myDA.DACOutput(TASK1_VOLTS);
        for (int i = 0; i < 80000 ; i++) {
                        count = count + 1;
                }
        myDA.DACOutput(0);
        myDA.Unlock();
        OSTimeDly(TICKS_PER_SECOND/20); // single tick delay is the smallest
possible sleep time
    }
}

/* Name: StartTask1
 * Description: Creates the task main loop.
 * Inputs: none
```

```
 * Outputs: none
 */
void StartTask2(void) {
    BYTE err = OS_NO_ERR;

    err = display_error( "StartTask2 fail:",
                    OSTaskCreatewName(  Task2Main,
                    (void *)NULL,
                    (void *) &DummyTask2Stk[USER_TASK_STK_SIZE],
                    (void *) &DummyTask2Stk[0],
                    DUMMYTASK2_PRIO, "Dummy Task 2" ));
}

/* Name: Task2Main
 * Description: NOP loop that can be timed
 *  Does nothing useful
 * Inputs:  void * pd -- pointer to generic data . Currently unused.
 * Outputs: none
 */
void    Task2Main( void * pd) {

    DWORD count = 0;

    while (1) {

        myDA.Lock();
        myDA.DACOutput(TASK2_VOLTS);
        for (int i = 0; i < 40000 ; i++) {
                    count = count + 1;
        }

        myDA.DACOutput(0);
        myDA.Unlock();
        OSTimeDly(TICKS_PER_SECOND/20);
    }
}
```

LCD.cpp

```
/*
```

```
 * LCD.cpp
 *
 *  Created on: Sep 22, 2016
 *      Author: Nancy Minderman
 *      nancy.minderman@ualberta.ca
 *      Much of the Initialization Code and raw
 *      font tables were taken from the Arduino code
 *      provided by Sparkfun
 *      https://www.sparkfun.com/products/10168
 *      Arduino sketch code
 *      http://playground.arduino.cc/Code/PCD8544
 */

#include "LCD.h"
#include "point.h"
#include <dspi.h>
#include <pins.h>
#include <basictypes.h>
#include <stdio.h>
#include <constants.h>
#include "error_wrapper.h"

#define __DEBUG_DSPI

LCD::LCD() {
    // TODO Auto-generated constructor stub
}
LCD::~LCD() {
    // TODO Auto-generated destructor stub
}
/* Initialize LCD hardware and DQSPI module*/
/* Name: Init
 * Description: Initializes both the SPI module on the MOD54415
 * and the LCD controller
 * Inputs: none
 * Outputs: none
 */
void LCD::Init(void) {

        init_spi();
        init_lcd();
}
```

```
/* Clear the display */
/* Name: Clear
 * Description: Clears the LCD by sending a screen full of space characters
 * Inputs: 48 X 84 pixels (504 Bytes) array of space characters.
clear_array is in bitmaps.h
 * Outputs: none
 */
void LCD::Clear(void){
        send_data((BYTE *)clear_array, SCREEN_SIZE);
}

/* Invert every pixel on screen */
/* Name: Invert
 * Description: Inverts evry pixel on screen from white to black and vice
versa
 * Note that the data in memory on the LCD does not change
 * Inputs: none
 * Outputs: none
 */
void LCD::Invert(void) {
    static BOOL inverted = false;
    if (inverted) {
        send_cmd(0x20);
        send_cmd(0x0C);

        inverted = false;
    } else {

        send_cmd(0x20);
        send_cmd(0x0D);

        inverted = true;
    }

}

/* Name: DrawBitmap
 * Description: Sends a bitmap of raw data to the screen starting at the
upper left hand pixel of the screen
 * Array must be 504 Bytes in size.
 * Inputs: const BYTE * bitmap
```

```
 * Outputs: none
 */
void LCD::DrawBitmap(const BYTE * bitmap) {
    Home();
    send_data((BYTE *) bitmap, SCREEN_SIZE);
    send_cmd(0x20);
    send_cmd(0x0C);
}

/* Name: DrawString
 * Description: Sends a c-style string to the display at the current index.
String must be null terminated
 * All elements in the string must be ASCII characters > space (0x20). Do
not send
 * non-printable characters. Bad things will happen.
 * Inputs: char * str - address of first character in string. String will
be displayed
 * at fixed locations on screen determined by points in ASCII_7 array in
bitmaps.h
 * Font table is also in bitmaps.h
 * Outputs: none
 */
void LCD::DrawString(char * str) {

    BYTE index = 0;

    while ( (*(str)) != '\0'){
        index = *str - ASCII_BASE; // convert ASCII character to index in
font table
        DrawChar((const BYTE *)&(ASCII_7[index]));
        str++;
    }
}
/* Name: DrawChar
 * Description: Draws a 7 pixel (width) by 8 pixel (height) sprite at the
current location.
 * Use the font tables in bitmaps.h to select a sprite.
 * Inputs: const BYTE * ch -  an array of raw data that represents the
character.
 * Outputs:
 */
void LCD::DrawChar(const BYTE * ch) {
```

```
    send_data((BYTE *) ch, CHAR_SIZE);
    send_cmd(0x20);
    send_cmd(0x0C);
}
/* Name:DrawChar
 * Description: Draws a 7 pixel(width) by 8 pixel( height) sprite at the
provided location
 * Inputs:  const BYTE * ch -  an array of raw data that represents the
character.
 *           point loc must be one of the locations in the char_index array
in bitmaps.h
 * Outputs: none
 */
void LCD::DrawChar(const BYTE * ch, point loc){
    move(loc);
    DrawChar(ch);
}

/* * Name: DrawBarGraph
 * Description: Displays a 7 X 8 box sprite to on the designated line
 * Inputs: BYTE line (valid from 0 - 5) BYTE length (valid from 0-11)
 * Outputs: none
 * Sprite is in bitmaps.h at index 96 ASCII array.
 */
void LCD::DrawBarGraph(BYTE line, BYTE length){
    BYTE sprite[] = {0xAA, 0x55,0xAA, 0x55,0xAA, 0x55,0xAA};
    BYTE start[] = {0, 12,24, 36,48, 60};
    while(length>0){
     DrawChar(sprite, char_index[start[line]]);
     DrawChar(sprite, char_index[start[line] + length]);
     length--;
    }


}

/* * Name: Test LCD
 * Description: Sends a fixed bitmap to the screen 84 X 48 pixels = 504
Bytes
 * Inputs: none
 * Outputs: none
 */
```

```
void LCD::TestLCD(void){
    Home();
    send_data((BYTE *) xkcdSandwich, SCREEN_SIZE);
}
    /* Name: Home
     * Description: Move the internal counter of the LCD to the upper
     * left hand point of the LCD
     * Inputs: none
     * Outputs: none
     */
void LCD::Home(void) {
    point origin = char_index[LINE1_ORIGIN];
    move(origin);
}
/* Name: Move
 * Description: Moves te internal counter of the LCD to the lcoation
specified
 * To write characters linmit the values of point to the members of the
char_index array
 * in bitmaps.h. No checks are made. If you overflow the x and y counters
you will get unexpected
 * results.
 * Inputs: point loc - location to move to
 * Outputs: none
 */
void LCD::Move(point loc) {
    move(loc);
}
/* Name: init_spi
 * Description: Initializes the SPI module on the 54415 to match the
requirements of the
 * NOKIA 51150 with PCD8544 LCD controller
 * Inputs: none
 * Outputs: none
 */
void LCD::init_spi(void){

        LCD_CLOCK.function(PINJ2_25_DSPI1_SCK);     // set SPI clock
function on pin J2[25]
        LCD_DATA_OUT.function(PINJ2_28_DSPI1_SOUT); // set SPI Data out
function on pin J2[28]
        LCD_RESET.function(PINJ2_26_GPIO);          // use this as a
```

```
programmable reset for the LCD reset
        LCD_C_D_LINE.function(PINJ2_27_GPIO);       // use this as the D/C
line 0 = command, 1 = data

        display_error("LCD::init_spi SemInit\n", OSSemInit(& DSPI_SEM, 0)
);                  // Use sem to be notified when transfer is finished.
         /* Initialize DSPI options
            void DSPIInit( BYTE SPIModule = DEFAULT_DSPI_MODLE, DWORD
Baudrate = 2000000,
                        BYTE QueueBitSize = 0x8, BYTE CS = 0x0F,
                        BYTE CSPol = 0x0F, BYTE ClkPolarity = 0x0,
                        BYTE ClkPhase = 0x1, BOOL DoutHiz = TRUE,
                        BYTE csToClockDelay = 0, BYTE delayAfterTransfer =
0 ) */
        DSPIInit(DEFAULT_DSPI_MODULE, // DEFAULT is SPI 1 so OK
                2000000,// Speed
                8,      // Queue bit size = 8 bits
                0x0,    // CS set these to all off. No chip selects please.
                0x1,    // CS Pol 1 = when CS is inactive, pin is pulled
high
                0,      // Clock polarity logic level when inactive. Set
this to 0
                0,      // Clock phase to 0 means data is captured on the
rising (or leading) edge of clock
                FALSE,   // DoutHiz should be true if the DOUT line needs
to be in high impedance in between transfers. Both work in our case.
                0,      // Use default for now: -QCD is a value in the
QDLYR register and will change the delay between the assertion of
                        //the chip select and the start of the DSPI clock.
Default setting of one half DSPI clk
                        // will be used if parameter is specified as 0x0 or
not included.
                0       // Use default for now: DTL is a value in the QDLYR
register and will change the delay following a transfer of
                        //a single WORD in the DSPI queue.  Default reset
value of 17/(fsys/2) will be used if
                        //parameter is specified as 0x0 or not included.
                );
}
/* Name: send_data
 * Description: Sends data to the LCD, with size being the total number of
bytes of data.
```

```
 * This will display it at the current location
 * Inputs: const Byte * data - data to be sent to screen in raw form. size
- number of bytes to display
 * Outputs: none
 */
void LCD::send_data(const BYTE * data, WORD size) {

    LCD_C_D_LINE = 1; // 1 = data

    DSPIStart(DEFAULT_DSPI_MODULE,(BYTE *) data,
                NULL, size, &DSPI_SEM); // send data via SPI bus
    display_error("LCD::send_data \n", OSSemPend( &DSPI_SEM, WAIT_FOREVER
));
}
/* Name: send_cmd
 * Description: Sends a command to the LCD. All possible commands are in
 * lcd.h
 * Inputs: BYTE command - valid LCD command from LCD.h
 * Outputs: none
 */
void LCD::send_cmd(BYTE command){


    LCD_C_D_LINE = 0;  // 0 = command

    // send command via the SPI bus
    //  commands are exactly 1 byte in size
    DSPIStart(DEFAULT_DSPI_MODULE, &command,
            NULL, 1, &DSPI_SEM);
    display_error("LCD::send_command \n", OSSemPend( &DSPI_SEM,
WAIT_FOREVER ));
}


/* Name: init_lcd
 * Description: See cite above for sparkfun and arduino info. This method
 * sends initialization commands to the PCD 8544 LCD controller
 * Inputs: none
 * Outputs: none
 */
void LCD::init_lcd(void) {
```

```
    LCD_RESET = 0;
    OSTimeDly(1); // minimal possible delay
    LCD_RESET = 1;
    // Insert your ex 2 code modifications here
    // H = 1
    send_cmd(0x21);
    send_cmd(0xB0);
    send_cmd(0x04);
    send_cmd(0x14);
    // H = 0
    send_cmd(0x20);
    send_cmd(0x0C);
    // End ex 2 modifications
    move(char_index[LINE1_ORIGIN]);
}
/* Name: move
 * Description: Set the internal counters of the LCDs to the location
provided
 * non-ASCII character data:
 * point.x should be between 0 and 83
 * point.y should be between 0 and 5
 * ASCII character data:
 * point should be one of the char_index points in char_index in bitmaps.h
 * Inputs: point location - new location for internal location counters of
LCD controller
 * Outputs: none
 */
void LCD::move(point loc) {
    send_cmd(0x20);
    send_cmd(0x40+loc.row);
    send_cmd(0x80+loc.col);
}
```

AD.cpp

```
/*
 * AD.cpp
 *
```

```
 *  Created on: Feb 9, 2017
 *      Author: nem1
 */

#include "AD.h"
#include <basictypes.h>
#include <sim.h>

AD::AD() {
    // TODO Auto-generated constructor stub

}

AD::~AD() {
    // TODO Auto-generated destructor stub
}

/* Name:Init
* Description: Initializes the entire register map
* of the AD converters
* Inputs: none
* Outputs: none
*/
void AD::Init(void){

       volatile WORD vw;

       //See MCF5441X RM Chapter 29
       sim2.adc.cr1 = 0;
       sim2.adc.cr2 = 0;
       sim2.adc.zccr = 0;
       sim2.adc.lst1 = 0x3210; //Ch 0....
       sim2.adc.lst2 = 0x7654; //ch 7 in result 0..7
       sim2.adc.sdis = 0; //All channels enabled
       sim2.adc.sr = 0xFFFF;
       for (int i = 0; i < 8; i++)
       {
          vw = sim2.adc.rslt[i];
          sim2.adc.ofs[i] = 0;
       }

       sim2.adc.lsr = 0xFFFF;
```

```
        sim2.adc.zcsr = 0xFFFF;

        sim2.adc.pwr = 0; //Everything is turned on
        sim2.adc.cal = 0x0000;
        sim2.adc.pwr2 = 0x0005;
        sim2.adc.div = 0x505;
        sim2.adc.asdiv = 0x13;
}

/* Name:StartAD
* Description: Starts the AD conversion. Only a single sample is
* converted
* Inputs: none
* Outputs: none
*/
void AD::StartAD(void){
     // Registers to modify for ex2
    // clear
    //   sim2.adc.sr
    //   sim2.adc.cr1
    sim2.adc.sr = 0x0000;
    sim2.adc.cr1 = ADC_CR1_START|ADC_CR1_LOOP;// start and run sequential
loop
}
/* Name:StopAD
* Description: Turn off the AD converter
* Inputs: none
* Outputs: none
*/
void AD::StopAD(void)
{
    // Registers to modify for ex2
    //   sim2.adc.cr1 ;
    sim2.adc.cr1 = ADC_CR1_STOP;
}
/* Name:ADDone
* Description: This method checks the status of
* the AD conversion.
* Inputs: none
* Outputs: Should return a boolean: true if the AD
* conversion is done and false if not done
*/
```

```
bool AD::ADDone(void)
{
    // Register to check for ex2
    //  sim2.adc.sr
    return sim2.adc.sr && 1; // checking 0x0000  and  1 ==> false,  0x0001
and  1 ==> true
}
/* Name: GetADResult
* Description: Reads the result of the AD conversion once
* the status register indicates that the conversion is done is ADDone
* Inputs: none
* Outputs: none
*/
WORD AD::GetADResult(int ch) //Get the AD Result
{
    // Register to modify for ex2
    // sim2.adc.rslt[ch]
    return sim2.adc.rslt[ch]>>3;
}
```

AD.cpp

```
/*
 * DA.cpp
 *
 *  Created on: Feb 9, 2017
 *      Author: nem1
 */

#include "DA.h"
#include <sim.h>
#include <stdio.h>
#include <iosys.h>

DA::DA() {
    // TODO Auto-generated constructor stub

}

DA::~DA() {
    // TODO Auto-generated destructor stub
```

```
}

/* Name:Init
* Description: Initializes the DA converters
* Inputs: none
* Outputs: none
*/
void DA::Init(void){

    BYTE err = OS_NO_ERR;
    err = display_error("DA::Init Sem", OSSemInit(&DA_lock_sem,1));

    sim1.ccm.dactsr = 0x0;         // DAC trigger select register in CCM
    sim2.adc.cal = ADC_CAL_DAC0;   // Selects the source of the ADCA3 input
as DAC0 output.
    sim2.dac[0].data = 0;          // Init data value to 0

    sim1.ccm.misccr2 &= ~(MISCCR2_ADC3EN);    // Disable ADC3 function on
J2.9
    sim1.ccm.misccr2 |= MISCCR2_DAC0SEL;      // Enable DAC0 output drive on
J2.9/Pin 4 of the
                                              // AD/DA 10 pin connector

    sim2.dac[0].cr &= ~DAC_CR_PDN;            // Enable DAC0 output

}

/* Name:DACOutput
* Description: Method for sending output to the DA converter.
* Use DAC0
* Inputs: none
* Outputs: none
*/
void DA::DACOutput(float volts)
{
    // Modify the data register to send data out
    // Max unsigned 12 bit number
    unsigned int X = (volts/3.3)*4096;
    sim2.dac[0].data = X ;

}
```

```
/* Name:Lock
* Description: Manages resource sharing of DAC0 to make sure
* only one task can use the DAC at any time. Call lock to make use of the
* DA converter
* Call Unlock once the DAC has finished writing.
* Inputs: none
* Outputs: none
*/
void DA::Lock(void){
    display_error("DA::Lock ", OSSemPend(&DA_lock_sem, WAIT_FOREVER));
}

/* Name:Unlock
* Description: Manages resource sharing of DAC0 to make sure
* only one task can use the DAC at any time. Call lock to make use of the
* DA converter
* Call Unlock once the DAC has finished writing.
* Inputs: none
* Outputs: none
*/
void DA::Unlock(void){
    display_error("DA::Unlock ", OSSemPost(&DA_lock_sem));
}
```