	EMBRY-RIDDLE AERONAUTICAL UNIVERSITY **Department of Electrical, Computer, and Software Engineering** **CEC322**: Microprocessor Laboratory (Spring 2019), **SECTION** PC51 Due Date: 5-FEB-2019 Turned in: 5-FEB-2019		
	PREPARED BY:	**EQUIPMENT S/N:**	**PERFORMED:**
	Gray, Andrea Pietz, Daniel	DK-TM4C123G BOARD-34	29-JAN-2019

LABORATORY #3

Introduction to Analog Signal Acquisition and Display

OBJECTIVES

- Increase knowledge and experience with the DK-TM4C123G Development Kit
- Introduce Analog to Digital conversion
- Utilize analog functions and constants in TivaWare© for the first time
- Continued use of the 'C' programming language

PURPOSE

To combine the implementations of the previous laboratory exercises and the use of the ADC (Analog to Digital Converter) on the TivaWare© TM4C123G Development Kit. The menu option in the program will allow the user to interact with the ADC in addition to the previous menu options from the prior exercises. Specifically, the menu driven software is required to enable the operation and testing of the peripheral for the ADC and make changes to the OLED display.

SPECIFICATIONS

For this laboratorial exercise it is required to create a working 'C' software project that runs on the DK-TM4C123G Development Kit. This program will require that two analog signal inputs to be read and their corresponding values displayed numerically and graphically onto the OLED. Similar to the previous projects, the virtual COM via USB will be used for UART0 and this exercise will be conduced through a menu-driven program.

The specific program requirements are listed below:

- Team specific "splash" screen on the OLED for no less than 2 seconds
- Three analog inputs displayed to the OLED
  a. Ain4-Ain7
- Each channel shall be displayed numerically, graphically, or not at all
- Each channel must be able to be controlled individually
- Toggle LED "heartbeat"
- Print menu of keypress options via keypress
- Quit the menu via keypress

PROCEDURE

There is a suggested procedure to follow, but it is viewed more as a guideline than an expected agenda. The procedure flow-chart that this exercise was based off is shown in Figure 1.0.

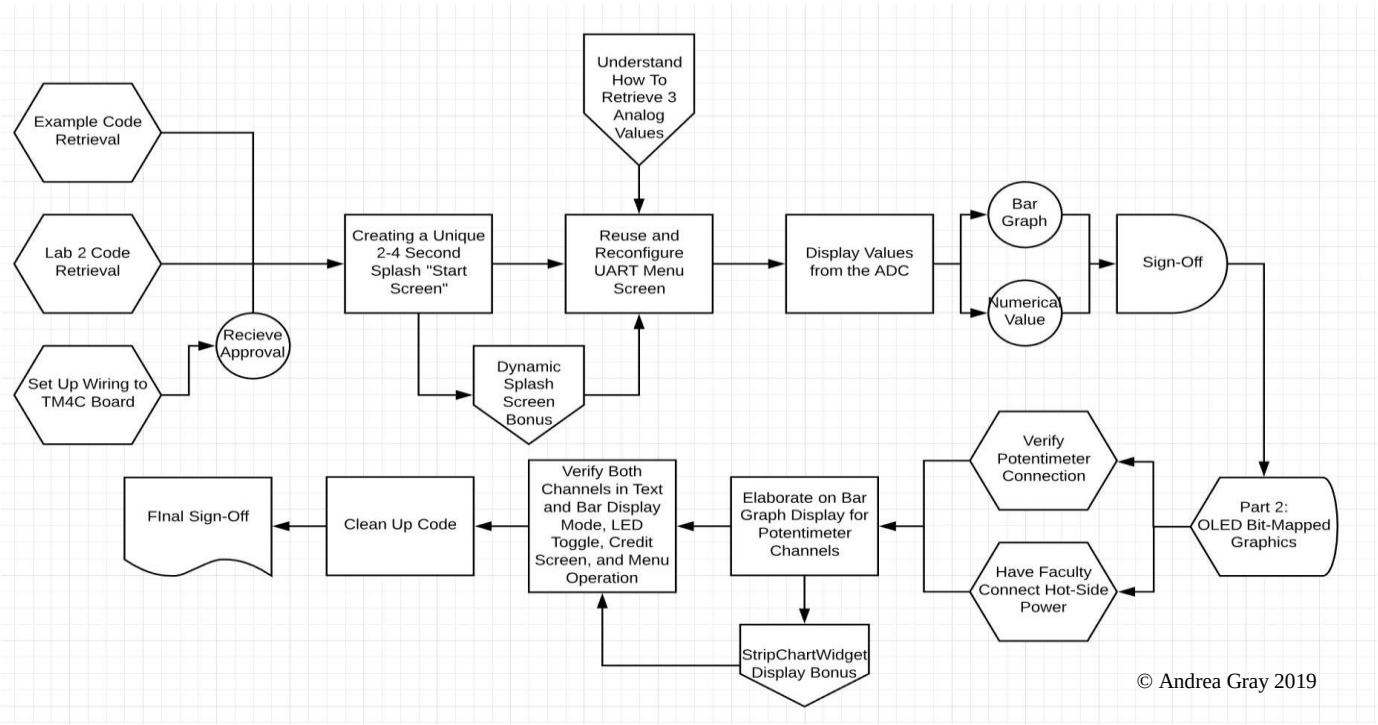


Figure 1.0 : Expected Lab Procedure

Process:

The process should begin with the gathering of the necessary source documents. These documents are the second laboratory exercise (UART Lab Menu project), Programming Example 4.3 in the TivaWare® Peripheral Driver Library, and TivaWare® peripheral example (single_ended.c). Once these are gathered and the COM port relates to the specifications used for all previous laboratory exercises, the source code from laboratory exercise two folder should be copied and opened to be the base code for this program with the file names being updated accordingly.

The set-up procedure is not new, but there was a specific variable name that needed to be changed in the driver library because it produced a linker error. This variable's error is not specified in the laboratory three manual and was very time consuming to identify and repair.

The first focus after the repair of the linker error is to implement a working "splash" screen. This screen should come up at the beginning of the program and should remain on the screen for no less than two seconds. This screen can simply be the output of a team name, but for extra credit a dynamic screen can be incorporated. A dynamic screen of a ball bouncing in different colors followed by the ball pulling the main OLED screen into view was performed for this development project.

The implementation of requirements for this program were not carried out in the order suggested in the example manual. The code for the menu, OLED output, and analog input channels were all completed before the final wiring of the board was done. The code, shown in the appendices below, describes the constants and blocks of code used from the source code in the TivaWare® peripheral example and which was created for this project. The basics of the menu function, the toggles, COM communication, OLED display, and LED "heartbeat" were maintained from previous exercises.

On the menu the new key press options are the number presses 1, 2, and 3 for toggling the display mode of each analog input respectively and independently. The graphical display for the analog inputs is a straightforward horizontal bar graph design. Each bar in the bar graph has a unique color for its respective analog input. These

graphical displays are also not affected by the 'Party Mode' function in this program to reduce value confusion since they are color coordinated.

A setback with the analog input display to the OLED is that while the graph's values extend to $(2^{12} - 1)$ the numerical display of the input needs to be in between the values [1-96]. This may not be an obvious change and did cause the need for some revision during check offs.

To understand how to receive the ADC0 analog values it is *necessary* to read through Application Note AN01247 [2]. This document, in the TivaWare® PDL, shows how to use the code sourced from single_ended.c to enable the ADC0 peripheral and associated pints and retrieve analog values from three channels (Ain4-Ain7 were suggested and are used). Subsequently, the values inputted from the analog channels should and are read and refreshed each loop iteration of the program. It was very beneficial to reference the code used for the button press counter in laboratory exercise two for the code development of the analog input values.

The first sign off for the exercise comes when the program can successfully input, convert, and display one analog input. As mentioned above, the procedure here was not following the specific flow chart displayed in Figure 1.0. The code was able to input, convert, and display all three analog inputs after code development efforts all at once.

With the code ready and debugged, it is time to create the board wiring with the potentiometer inputs in the TM4C123G board. The board wiring, since it can prove detrimental if not done correctly, had to be examined by a laboratory adviser before making the hot 3.3V connection. The wiring is shown below in Appendix B. As shown, three potentiometers are used for the three individual analog connections used. Each potentiometer used has three connections; one for each end of the resistor (GND and +3.3V) and one for the sliding wiper. The sliding wiper is connected to the respective analog input to regulate the input as shown in the OLED numerical or graphical display.

Once the wiring was approved and connected, the entire program was compiled, executed, demonstrated, and signed off as shown in Appendix C.

REPORT DISCUSSION

1. A list of the TivaWare® functions and constants that were used for the first time are listed below (all using #include driverlib/adc.h):
   a. GIOPinTypeADC( );
   b. ADCSequenceConfigure( );
   c. ADCSequenceStepConfigure( );
   d. ADCSequenceEnable( );
   e. ADCProcessorConfigure( );
   f. ADCIntStatus( );
   g. ADCIntClear( );
   h. ADCSequenceDataGet( );
   i. SYSCTL_PERIPH_ADC0
   j. ADC0_BASE
   k. ADC_TRIGGER_PROCESSOR
   l. ADC_CTL_CH4
   m. ADC_CTL_CH5
   n. ADC_CTL_CH6
   o. ADC_CTL_IE
   p. ADC_CTL_END

2. The sequence used within ADC0 for this lab was sequence 1. We chose this because this sequence allows for 4 possible step implications. This sequence is the most appropriate sequence because the project uses 3 steps within the sequence. The ADC sequences 1 and 4 have FIFO depths of 4 which is the most appropriate for this development. On the other hand, sequence 3 has a FIFO depth of 1, which is not enough, and sequence 0 has a FIFO depth of 8, which is too much.

3. Some motivations for why the designers of the TivaWare® ADC chose to have 4 ADC sequences defined, each with their own distinct number of steps and independent triggers, are:
   a. Readability
   b. Efficiency of searching
   c. Timing of code processing

4. The most common types of ADC and their description and operations are:
    a. Flash ADC
        i. Ladder voltage comparison which decreases time but increases transistor count.
    b. Successive Approximation ADC
        i. Using a binary search and sorting pattern to converge on the given voltage. This type of ADC is slower in timing, but the transistor count required for implementation is decreased in comparison to Flash ADC.
    c. Ramp Compare ADC
        i. This ADC has a fixed precision and if the given voltage is above the current convergence cycle location then the guess counter increases by one predetermined precision point. This ADC type is the slowest of the three, but it requires the least amount of hardware to physically implement.

5. In lines {850-851} of the GrStringDraw function in the TivaWare® file string.c, the ASSERT( ) function parameter pContext's existence is validated and then, on line 851, the function checks if the pfnStringRenderer member of the pContext structure exists. If it does not, then an exception is thrown.

6. Assuming a 1.65 V signal is applied to Ain4, the output of the ADC will be 2,253 and 0x8CD considering that the 12-bit ADC on the DK-TM4C123G uses a 0.0 V to 3.0 V operating range.

7. Some of the surprises that our team encountered were the ease of splash screen development and the potentiometer running smoother than expected. The splash screen was actually very fun to develop and the reward of seeing the splash screen up and running was a huge boost to confidence and motivation in the laboratory exercise. Additionally, the realization of the fact that the code executes at a substantially faster pace when the LED 'heartbeat' is toggled of was, although very logical, very surprising. The exercise did not come without its share of challenges though. Some challenges throughout the lab was the fact that our team made the code all at once, skipping the individual signoffs and offered procedural steps, with some of the main features, such as the splash screen, in separate files. This developmental strategy allowed for an easier assimilation of ADC functions and outputs, but it proved to be more difficult in integration of the program's key features. Once the splash screen, ADC functions, and other necessary components were all in cooperation, the exercise itself was not troublesome. If this lab were to be done again, I do not think our team would stray from our developmental procedure, but we would make sure to integrate all portions and functions of code before any further development is conduced. In future developments of this exercise with different teams, the only suggestion that can be made is more documentation and explanation on how to fix linker errors in the laboratory exercise manual itself.

CONCLUSION

With experience comes wisdom, with knowledge comes fulfillment, and in the attainment of those comes the fire to continually drive the process. Plutarch, a Greek philosopher, stated, "The mind is not a vessel that needs filling, but a wood that needs igniting." [4]. In the increased usage of the TM4C123G Development Kit, the fire of understanding the connection and unification of hardware and software is not only ignited, but furthermore it is kindled. The vast applications of the TivaWare® Development Kit allows for the user to not only implement interesting procedures, but the laboratory process itself indulges the user in the gained knowledge and experience in the practical utilization of the provided software and hardware correlation. The expansion of the ADC functionalities and peripheral usage on the TivaWare® foundation, set up in the first laboratory exercise, is the growth of the fire of intelligence in each mind. The quote by Plutarch explains that knowledge is not a finite data set, instead knowledge is the infinite accumulation of experience and intellect. While the roots of education can be bitter, as Aristotle quotes, the fruit is sweet. The laboratory exercises, while meticulous and trying, set aflame a driven passion that only grows with each saccharine taste of the fruits of one's labor.

APPENDICES:

APPENDIX A: Lab Code

```
//*****************************************************************************
//
// CS322.50 Labratory 2 Software File
// Developed by: Andrea Gray and Daniel Pietz (c)
// Version: 1.40 28-JAN-2019
//
//*****************************************************************************

//*****************************************************************************
//
// uart_echo.c - Example source for reading data from and writing data to the
// UART in an interrupt driven fashion for Labratory Exercise 2.
//
// Copyright (c) 2011-2017 Texas Instruments Incorporated.
//
// This is part of revision 2.1.4.178 of the DK-TM4C123G Firmware Package.
//
//*****************************************************************************

#include <stdint.h>                             // Standard library header for
                                                // integers with varying widths
#include <stdio.h>                              // Standard library header for
                                                // input and outputs -- sprintf
                                                // in this program specifically
#include "driverlib/adc.h"                      // ADC driver files in the
                                                // driver library
#include "utils/uartstdio.h"                    // Header file for driver files
                                                // for UART I/O
#include <stdbool.h>                            // Standard library header for
                                                // boolean data types
#include <time.h>                               // Header file with four main
                                                // variable types for
                                                // manipulating date and time
                                                // information
#include <stdlib.h>                             // Standard C library header --
                                                // malloc in this program
                                                // specifically
#include <string.h>                             // Header file for the use and
                                                // manipulation of strings
#include "inc/hw_memmap.h"                      // Header file for BASE call use
#include "driverlib/debug.h"                    // TM4C123G debugging header
                                                // file
#include "driverlib/gpio.h"                     // Header file for all GPIO
                                                // function calls
#include "driverlib/sysctl.h"                   // Header file for System
                                                // Control Specs
#include "driverlib/uart.h"                     // Header file for UART function
                                                // calls
#include "grlib/grlib.h"                        // Header file for output calls
#include "drivers/cfal96x64x16.h"               // Header file for OLED display
                                                // dimension specifications
#include "drivers/buttons.h"                    // Header file for push-buttons
                                                // counter
#define LEDon 20000                             // defines the on period of the
                                                // LED in ms
#define LEDoff 380000                           // defines the off period of the
                                                // LED in ms
#define refreshRate 60000                       // The refresh rate for splash
                                                // screen output

// ADC data display type
typedef enum {off, numeric, histogram, terminator} displayType;

//*****************************************************************************
//
// Function Declarations
//
//*****************************************************************************
void putString(char *str);
void clear();
void printMenu();
void blinky(volatile uint32_t ui32Loop);
void InitConsole(void);

//*****************************************************************************
//
```

```
77  // This example application utilises the UART to echo text. All characters
78  // recoeved on the UART are transmitted back to the UART.
79  //
80  //*****************************************************************************
81
82  //*****************************************************************************
83  //
84  // The error routine that is called if the driver library encounters an error.
85  //
86  //*****************************************************************************
87  #ifdef DEBUG
88  void __error__(char *pcFilename, uint32_t ui32Line) {
89  }
90  #endif
91
92  //*****************************************************************************
93  //
94  // Holds the current, debounced state of each button.  0 = pressed.
95  // We assume that we start with all the buttons released (though if one is
96  // pressed when the application starts, this will be detected).
97  //
98  //*****************************************************************************
99  static uint8_t g_ui8ButtonStates = ALL_BUTTONS;
100
101 //*****************************************************************************
102 //
103 // Initializes the GPIO pins used by the board pushbuttons with a weak
104 // pull-up.
105 //
106 //*****************************************************************************
107 void ButtonsInit(void) {
108   //
109   // Enable the GPIO port to which the pushbuttons are connected.
110   //
111   SysCtlPeripheralEnable(BUTTONS_GPIO_PERIPH);
112
113   //
114   // Set each of the button GPIO pins as an input with a pull-up.
115   //
116   GPIODirModeSet(BUTTONS_GPIO_BASE, ALL_BUTTONS, GPIO_DIR_MODE_IN);
117   GPIOPadConfigSet(BUTTONS_GPIO_BASE, ALL_BUTTONS,
118                    GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
119
120   //
121   // Initialize the debounced button state with the current state read from
122   // the GPIO bank.
123   //
124   g_ui8ButtonStates = GPIOPinRead(BUTTONS_GPIO_BASE, ALL_BUTTONS);
125 }
126
127 //*****************************************************************************
128 //
129 // Send a string to the UART.
130 //
131 //*****************************************************************************
132 void UARTSend(const uint8_t *pui8Buffer, uint32_t ui32Count) {
133   //
134   // Loop while there are more characters to send.
135   //
136   while(ui32Count--) {
137     //
138     // Write the next character to the UART.
139     //
140     UARTCharPutNonBlocking(UART0_BASE, *pui8Buffer++);
141   }
142 }
143
144 //*****************************************************************************
145 //
146 // The main function to intialize the UART, LED, OLED, and run through the
147 // program
148 //
149 //*****************************************************************************
150
```

```
int main(void) {
    tRectangle sRect;                           // OLED rectangle variable
    tContext sContext;                          // OLED graphics buffer
    int gNumCharRecv = 0;                       // Count for characters
                                                // recieved by the UART
    volatile uint32_t ui32Loop;                 // Blinky LED volatile
                                                // loop.
    bool shouldFlood = false;                   // "Flood" character toggle
    bool shouldBlink = true;                    // LED blinky toggle

    // positional information useed to animate the splash screen
    int16_t xValLast = 0;
    int16_t yValLast = 38;
    int16_t xVal = 0;
    int32_t yVal = 38;

    volatile int16_t val[3];                     // volatile variable to store
                                                // potentiometer data
    volatile uint32_t uiLoop;                    // refresh rate increment
                                                // variable

    //*****************************************************************************
    // Counter variables
    //*****************************************************************************
    int tickCount = 0;                          // Clock ticks for flood
                                                // character.
    int whileLoop = 1;                          // Looping through the main
                                                // infinite while loop
    int colorSwitch = 0;                        // OLED Color toggle
    int shouldCycle = 0;                        // Boolean for Party Mode
    uint32_t buttonState = 0;                   // Boolean to check if a
                                                // button was pressed.
    int buttonCounter = 0;                      // Counting how many times
                                                // a button was pressed.
    int lastPressed = 0;                        // Saves the number of the
                                                // last button pressed for the
                                                // OLED output.

    //*****************************************************************************
    //
    // Configure ADC0 for a single-ended input and a single sample.  Once the
    // sample is ready, an interrupt flag will be set.  Using a polling method,
    // the data will be read then displayed on the console via UART0.
    //
    //*****************************************************************************

    // This array is used for storing the data read from the ADC FIFO. This
    // project uses sequence 1 which has a FIFO depth of 4.
    uint32_t pui32ADC0Value[4];

    // For this example ADC0 is used with AIN0 on port E7.
    // GPIO port D needs to be enabled so these pins can be used.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

    // Selecting the analog ADC function for pins 4 5 and 6.
    GPIOPinTypeADC(GPIO_PORTD_BASE, GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6);

    // Enable sequence 1 with a processor signal trigger.  Sequence 1
    // will do a single sample when the processor sends a signal to start the
    // conversion.
    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);

    // Configure step 0, 1, and 2 on sequence 1.  Sample channels 4, 5, and 6 in
    // single-ended mode (default). Tell the ADC logic that channel 6 is the
    // last conversion on sequence 1 (ADC_CTL_END).  Sequence 1 has 4 steps.
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_CH4);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_CH5);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_CH6 | ADC_CTL_IE |
                             ADC_CTL_END);

    // Since sample sequence 1 is now configured, it must be enabled.
    ADCSequenceEnable(ADC0_BASE, 1);

    //*****************************************************************************
    //
```

```
227     // Set the clocking to run directly from the crystal.
228     //
229     //*****************************************************************************
230     SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
231                    SYSCTL_XTAL_16MHZ);
232
233     //*****************************************************************************
234     //
235     // Enable the GPIO port that is used for the on-board LED.
236     //
237     //*****************************************************************************
238     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
239
240     //*****************************************************************************
241     //
242     // Check if the LED peripheral access is enabled. If it is not, wait inside
243     // the loop until it is.
244     //
245     //*****************************************************************************
246     while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOG)) {
247     }
248
249     //*****************************************************************************
250     //
251     // Enable the GPIO pin 2 for the LED.  Set the direction as output, and
252     // enable the GPIO pin for digital function.
253     //
254     //*****************************************************************************
255     GPIOPinTypeGPIOOutput(GPIO_PORTG_BASE, GPIO_PIN_2);
256
257
258     //*****************************************************************************
259     //
260     // Initialise the OLED display driver.
261     //
262     //*****************************************************************************
263     CFAL96x64x16Init();
264
265     //*****************************************************************************
266     //
267     // Initialise the OLED graphics context.
268     //
269     //*****************************************************************************
270     GrContextInit(&sContext, &g_sCFAL96x64x16);
271
272
273     //*****************************************************************************
274     //
275     // Enable the peripherals used by this example.
276     //
277     //*****************************************************************************
278     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);  // GPIO Pin Set
279     SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);  // UART0 Set
280
281
282     //*****************************************************************************
283     //
284     // Set GPIO 0 and 1 as UART pins.
285     //
286     //*****************************************************************************
287     GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
288
289     //*****************************************************************************
290     //
291     // Configure the UART for custom 115,200 baud rate.
292     //
293     //*****************************************************************************
294     UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
295                         (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
296                          UART_CONFIG_PAR_NONE));
297
298
299     //*****************************************************************************
300     //
301     // Check to see if the GPIO peripheral is ready. If it is not, wait inside
```

```
    // the loop until it is.
    //
    //*****************************************************************************
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOG)) {
    }

    //*****************************************************************************
    //
    // Set the LED Pin output to pin 2.
    //
    //*****************************************************************************
    GPIOPinTypeGPIOOutput(GPIO_PORTG_BASE, GPIO_PIN_2);

    ButtonsInit();                          // Enable buttons for
                                            // each loop iteration.
    int Looper = 0;                         // Set loop variable and
                                            // initialize to 0.
    clear();                                // calling the clear function
                                            // to clear any remaining
                                            // characters in text window
    printMenu();                            // calling the print function
                                            // for outputting the menu
                                            // to the interface screen

    // Setting the OLED window boundaires
    sRect.i16XMin = 0;
    sRect.i16YMin = 0;
    sRect.i16XMax = GrContextDpyWidthGet(&sContext);
    sRect.i16YMax = GrContextDpyHeightGet(&sContext);
    GrContextForegroundSet(&sContext, ClrBlack);  // Set the banner color to
                                            // dark blue.
    GrRectFill(&sContext, &sRect);          // Output the dimensions and
                                            // fill the OLED with the
                                            // banner.
    int jumpTick = -1;                      // Keeps track of splash screen
                                            // ball location in cycle.
    int Rad = 5;                            // Radius of the ball in pixels.
    int color = 0;                          // Color variable for the ball.

    //*****************************************************************************
    //
    // Infinite While Loop
    //
    //*****************************************************************************
    while(1)
    {
      // Reading the state of the button.
      buttonState = (GPIOPinRead(BUTTONS_GPIO_BASE,ALL_BUTTONS));
      // The button codes are : 30-UP, 29-DOWN, 27-LEFT, 23-RIGHT, and 15-SELECT
      xVal+=2;
      if(jumpTick == -1) {
        jumpTick = 0;
      }

      if(jumpTick != -1) {
        yVal = 38 - (10*jumpTick-jumpTick*jumpTick);
        jumpTick++;
        if(jumpTick > 10) {
          jumpTick = -1;
        }
      }

      GrContextInit(&sContext, &g_sCFAL96x64x16); // Resets OLED output context.

      // Switch case for the color of an output to the OLED.
      // Key: 0-Blue, 1-Red, 2-Green, 3-Black
      switch(color) {
      case 0:
        GrContextForegroundSet(&sContext, ClrBlue);
        break;
      case 1:
        GrContextForegroundSet(&sContext, ClrRed);
        break;
      case 2:
```

```
            GrContextForegroundSet(&sContext, ClrGreen);
            break;
          case 3:
            GrContextForegroundSet(&sContext, ClrBlack);
            // Resetting the parameters of the OELD window
            sRect.i16XMin = 0;
            sRect.i16YMin = 0;
            sRect.i16XMax = xVal;
            sRect.i16YMax = GrContextDpyHeightGet(&sContext);
            GrRectFill(&sContext, &sRect);
            break;
          }

          // Filling in the splash screen ball OLED output
          GrCircleFill(&sContext, xValLast,yValLast, Rad);
          GrContextForegroundSet(&sContext, ClrWhite);
          GrCircleFill(&sContext, xVal,yVal, Rad);
          GrContextFontSet(&sContext, g_psFontCm12/*g_psFontFixed6x8*/);
          GrFlush(&sContext);
          xValLast = xVal;
          yValLast = yVal;
          if(xVal >= 106) {                         // If the ball comes to the
                                                    // edge of the OLED window
                                                    // parameters, then go back to
                                                    // start of screen and change
                                                    // ball color.
            xVal = -13;
            color++;
          }

          // If the ball reaches the last color then quit the splash screen.
          if(color == 4) {
            break;
          }

          // Waiting for refresh rate to be met so that the splash screen does not
          // go through too fast or too slow.
          for(uiLoop = 0; uiLoop < refreshRate; uiLoop++) {
          }
        }

        //*****************************************************************************
        //
        // Fill the top part of the screen in the parameters below with dark blue to
        // create the banner.
        //
        //*****************************************************************************
        sRect.i16XMin = 0;
        sRect.i16YMin = 0;
        sRect.i16XMax = GrContextDpyWidthGet(&sContext) - 1;
        sRect.i16YMax = 9;
        GrContextForegroundSet(&sContext, ClrDarkBlue);
        GrRectFill(&sContext, &sRect);

        //*****************************************************************************
        //
        // Change foreground for white text.
        //
        //*****************************************************************************
        GrContextForegroundSet(&sContext, ClrWhite);

        //*****************************************************************************
        //
        // Put the application name in the middle of the banner.
        //
        //*****************************************************************************
        GrContextFontSet(&sContext, g_psFontFixed6x8);
        GrStringDrawCentered(&sContext, "Gray & Pietz", -1,
                             GrContextDpyWidthGet(&sContext) / 2, 4, 0);

        //*****************************************************************************
        //
        // Initialization of potentiometer values and display types.
        //
        //*****************************************************************************
```

```
452     val[0] = 0;
453     val[1] = 0;
454     val[2] = 0;
455     displayType aDisp[3];
456     aDisp[0] = off;
457     aDisp[1] = off;
458     aDisp[2] = off;
459
460     //*****************************************************************************
461     //
462     // Main infinite while loop used for ADC and I/O
463     //
464     //*****************************************************************************
465     while(whileLoop != 0) {
466
467       ADCProcessorTrigger(ADC0_BASE,1);           // Triggers a read from the ADC.
468       while(!ADCIntStatus(ADC0_BASE,1,false)) {   // Waiting for an input.
469       }
470       ADCIntClear(ADC0_BASE,1);                   // Clear the input port.
471
472       // Getting the values from the port.
473       ADCSequenceDataGet(ADC0_BASE, 1, pui32ADC0Value);
474
475       // Variable incrementations for every loop iteration.
476       Looper++;
477       whileLoop++;
478       tickCount++;
479
480       //*****************************************************************************
481       //
482       // Setting the potentiometer values to display ADC values to fit in the
483       // range that the OLED can handle.
484       //
485       //*****************************************************************************
486       val[0] = ((96*pui32ADC0Value[0])/4095);
487       val[1] = ((96*pui32ADC0Value[1])/4095);
488       val[2] = ((96*pui32ADC0Value[2])/4095);
489
490       //*****************************************************************************
491       //
492       // Flood character output character and timing.
493       //
494       //*****************************************************************************
495       if(tickCount == 3) {
496         tickCount = 0;
497         if(shouldFlood == 1) {
498           UARTCharPut(UART0_BASE, '@');
499         }
500       }
501
502       //*****************************************************************************
503       //
504       // Blinky function toggle.
505       //
506       //*****************************************************************************
507       if(shouldBlink) {
508         blinky(ui32Loop);
509       }
510
511       //*****************************************************************************
512       //
513       // Checking for a button press, displaying output as designated, and
514       // incrementing counts accordingly.
515       //
516       //*****************************************************************************
517       buttonState = (~GPIOPinRead(BUTTONS_GPIO_BASE,ALL_BUTTONS) && ALL_BUTTONS);
518       if(GPIOPinRead(BUTTONS_GPIO_BASE, ALL_BUTTONS)) {
519         buttonCounter++;
520         // Key: 1-UP, 2-DOWN, 3-LEFT, 4-RIGHT, 5-SELECT
521         switch (buttonState) {
522         case UP_BUTTON:
523           lastPressed = 1;
524           break;
525         case DOWN_BUTTON:
526           lastPressed = 2;
```

```
527             break;
528           case LEFT_BUTTON:
529             lastPressed = 3;
530             break;
531           case RIGHT_BUTTON:
532             lastPressed = 4;
533             break;
534           case SELECT_BUTTON:
535             lastPressed = 5;
536             break;
537           }
538         }
539
540         //*****************************************************************************
541         //
542         // Changing the OLED output depending on Party Mode toggle and Banner Color
543         // menu selection.
544         //
545         //*****************************************************************************
546         if(shouldCycle == 1) {
547           if(colorSwitch == 2) {
548             colorSwitch = -1;
549           }
550
551           // Setting OLED display as full avaliable rectangle output.
552           colorSwitch++;
553           sRect.i16XMin = 0;
554           sRect.i16YMin = 0;
555           sRect.i16XMax = GrContextDpyWidthGet(&sContext) - 1;
556           sRect.i16YMax = GrContextDpyHeightGet(&sContext) - 1;
557
558           // Switch case to determine the next OLED color output.
559           switch (colorSwitch) {
560           case 0:
561             GrContextForegroundSet(&sContext, ClrDarkBlue);
562             GrContextBackgroundSet(&sContext, ClrDarkBlue);
563             break;
564           case 1:
565             GrContextForegroundSet(&sContext, ClrRed);
566             GrContextBackgroundSet(&sContext, ClrRed);
567             break;
568           case 2:
569             GrContextForegroundSet(&sContext, ClrGreen);
570             GrContextBackgroundSet(&sContext, ClrGreen);
571             break;
572           default:
573             GrContextForegroundSet(&sContext, ClrDarkBlue);
574             GrContextBackgroundSet(&sContext, ClrDarkBlue);
575             break;
576           }
577
578           // Re-draw the OLED rectangular output with above determined color
579           // and correct banner message.
580           GrRectFill(&sContext, &sRect);
581           GrContextForegroundSet(&sContext, ClrWhite);
582           GrContextFontSet(&sContext, g_psFontFixed6x8);
583           GrStringDrawCentered(&sContext, "Gray & Pietz", -1,
584                                GrContextDpyWidthGet(&sContext) / 2, 4, 0);
585         }                                       // end shouldCycle()
586
587         //*****************************************************************************
588         //
589         // While loop when a character is inputted in the PuTTY window.
590         //
591         //*****************************************************************************
592         while(UARTCharsAvail(UART0_BASE)) {
593           char str[50];                         // Empty string declaration for
594                                                 // sprintf functions.
595           int32_t local_char;                   // Setting the input character
596                                                 // to a local variable.
597
598           //*************************************************************************
599           //
600           // Re-draw the OLED with updated statistics
601           //
```

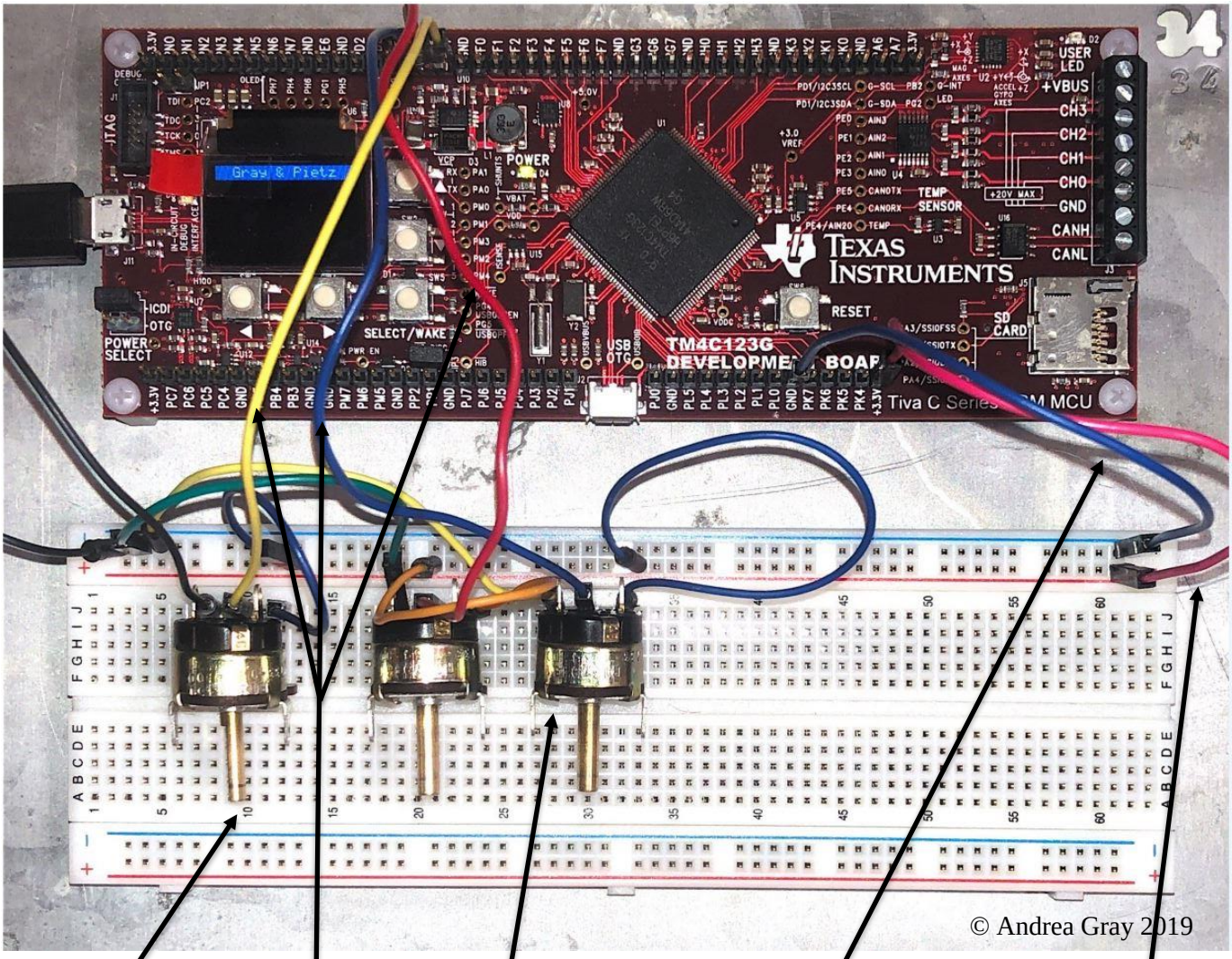
```
602         //*****************************************************************************
603         local_char = UARTCharGetNonBlocking(UART0_BASE);
604
605         //*****************************************************************************
606         //
607         // If the input character is not invalid, begin the character matching
608         // statment below through the switch statment and act accordingly to
609         // the user input.
610         //
611         // Key: 67 'C' - Banner Color Switch, 69 'E' - Clear Interface Window,
612         //      70 'F' - Flood Character Toggle, 76 'L' - LED Toggle,
613         //      77 'M' - Reprint Menu, 80 'P' - Party Mode, 81 'Q' - Quit Program
614         //
615         //*****************************************************************************
616         if (local_char != -1) {
617           gNumCharRecv++;                       // Character input counter
618           UARTCharPut(UART0_BASE, local_char);  // Sending a single character
619                                                 // through the UART_TX
620                                                 // (transmitting)channel
621
622           switch(local_char) {
623           case 67:
624             if(colorSwitch == 2) {
625               colorSwitch = -1;
626             }
627             colorSwitch++;                      // Color cycle counter
628
629             // Specifying the banner area and recloration for menu options
630             // C and P.
631             sRect.i16XMin = 0;
632             sRect.i16YMin = 0;
633             sRect.i16XMax = GrContextDpyWidthGet(&sContext) - 1;
634             sRect.i16YMax = 9;
635             switch (colorSwitch) {
636             case 0:
637               GrContextForegroundSet(&sContext, ClrDarkBlue);
638               GrContextBackgroundSet(&sContext, ClrDarkBlue);
639               break;
640             case 1:
641               GrContextForegroundSet(&sContext, ClrRed);
642               GrContextBackgroundSet(&sContext, ClrRed);
643               break;
644             case 2:
645               GrContextForegroundSet(&sContext, ClrGreen);
646               GrContextBackgroundSet(&sContext, ClrGreen);
647               break;
648             default:
649               GrContextForegroundSet(&sContext, ClrDarkBlue);
650               GrContextBackgroundSet(&sContext, ClrDarkBlue);
651               break;
652             }
653
654             // Filling in the rest of the OLED screen after banner is
655             // updated.
656             GrRectFill(&sContext, &sRect);
657             GrContextForegroundSet(&sContext, ClrWhite);
658             GrContextFontSet(&sContext, g_psFontFixed6x8);
659             GrStringDrawCentered(&sContext, "Gray & Pietz", -1,
660                                  GrContextDpyWidthGet(&sContext) / 2, 4, 0);
661             break;
662
663           case 69:
664             clear();
665             break;
666
667           case 70:
668             if(shouldFlood == 0) {
669               shouldFlood = 1;
670             }
671             else {
672               shouldFlood = 0;
673             }
674             break;
675
676           case 76:
```

```
677             if(shouldBlink == 0)
678               shouldBlink = 1;
679             else
680               shouldBlink = 0;
681             break;
682
683           case 77:
684             printMenu();
685             break;
686
687           case 80:
688             if(shouldCycle == 0) {
689               shouldCycle = 1;
690             }
691             else
692             {
693               shouldCycle = 0;
694             }
695             break;
696
697           case 81:
698             putString("\n\rBYE!");                // Goodbye message to CPU
699                                                    // window.
700
701             // Re-draw OLED with goodbye statment in red font.
702             sRect.i16XMin = 0;
703             sRect.i16YMin = 0;
704             sRect.i16XMax = GrContextDpyWidthGet(&sContext) - 1;
705             sRect.i16YMax = GrContextDpyHeightGet(&sContext) - 1;
706             GrContextForegroundSet(&sContext, ClrBlack);
707             GrContextBackgroundSet(&sContext, ClrBlack);
708             GrRectFill(&sContext, &sRect);
709             GrContextForegroundSet(&sContext, ClrRed);
710             GrContextFontSet(&sContext, g_psFontFixed6x8);
711             GrStringDrawCentered(&sContext, "Goodbye", -1, GrContextDpyWidthGet(&sContext) / 2, 30, false);
712             whileLoop = 0;
713             break;
714           case 49:
715             aDisp[0]++;
716             break;
717           case 50:
718             aDisp[1]++;
719             break;
720           case 51:
721             aDisp[2]++;
722             break;
723
724         }                                          // End menu switch statment.
725       }                                            // End valid character check.
726     }                                              // End character scan loop.
727
728     // Reset the value of each display type once it runs off the enum range.
729     if(aDisp[0] == terminator) aDisp[0] = off;
730     if(aDisp[1] == terminator) aDisp[1] = off;
731     if(aDisp[2] == terminator) aDisp[2] = off;
732     char tempStr[5];
733
734     // Switch statement for ADC output 0 as a number, graph, or not displayed.
735     switch(aDisp[0]) {
736     case off:
737       sRect.i16XMin = 0;
738       sRect.i16YMin = 16;
739       sRect.i16XMax = GrContextDpyWidthGet(&sContext);
740       sRect.i16YMax = 32;
741       GrContextForegroundSet(&sContext, ClrBlack);
742       GrRectFill(&sContext, &sRect);
743       break;
744     case numeric:
745       sRect.i16XMin = 0;
746       sRect.i16YMin = 16;
747       sRect.i16XMax = GrContextDpyWidthGet(&sContext);
748       sRect.i16YMax = 32;
749       GrContextForegroundSet(&sContext, ClrBlack);
750       GrRectFill(&sContext, &sRect);
751       GrContextForegroundSet(&sContext, ClrWhite);
```

```
752             sprintf(tempStr,"%i",pui32ADC0Value[0]);
753             GrStringDrawCentered(&sContext, tempStr, -1,
754                                  GrContextDpyWidthGet(&sContext) / 2, 24, 16);
755             break;
756         case histogram:
757             sRect.i16XMin = 0;
758             sRect.i16YMin = 16;
759             sRect.i16XMax = val[0];
760             sRect.i16YMax = 32;
761             GrContextForegroundSet(&sContext, ClrRed);
762             GrRectFill(&sContext, &sRect);
763             sRect.i16XMin = val[0];
764             sRect.i16XMax = 96;
765             GrContextForegroundSet(&sContext, ClrBlack);
766             GrRectFill(&sContext, &sRect);
767             break;
768         }
769
770         // Switch statment for ADC output 1 as a number, graph, or not displayed.
771         switch(aDisp[1]) {
772         case off:
773             sRect.i16XMin = 0;
774             sRect.i16YMin = 32;
775             sRect.i16XMax = GrContextDpyWidthGet(&sContext);
776             sRect.i16YMax = 48;
777             GrContextForegroundSet(&sContext, ClrBlack);
778             GrRectFill(&sContext, &sRect);
779             break;
780         case numeric:
781             sRect.i16XMin = 0;
782             sRect.i16YMin = 32;
783             sRect.i16XMax = GrContextDpyWidthGet(&sContext);
784             sRect.i16YMax = 48;
785             GrContextForegroundSet(&sContext, ClrBlack);
786             GrRectFill(&sContext, &sRect);
787             GrContextForegroundSet(&sContext, ClrWhite);
788             sprintf(tempStr,"%i",pui32ADC0Value[1]);
789             GrStringDrawCentered(&sContext, tempStr, -1,
790                                  GrContextDpyWidthGet(&sContext) / 2, 40, 16);
791             break;
792         case histogram:
793             sRect.i16XMin = 0;
794             sRect.i16YMin = 32;
795             sRect.i16XMax = val[1];
796             sRect.i16YMax = 48;
797             GrContextForegroundSet(&sContext, ClrGreen);
798             GrRectFill(&sContext, &sRect);
799             sRect.i16XMin = val[1];
800             sRect.i16XMax = 96;
801             GrContextForegroundSet(&sContext, ClrBlack);
802             GrRectFill(&sContext, &sRect);
803
804             break;
805         }
806
807         // Switch statment for ADC output 2 as a number, graph, or not displayed.
808         switch(aDisp[2]) {
809         case off:
810             sRect.i16XMin = 0;
811             sRect.i16YMin = 48;
812             sRect.i16XMax = GrContextDpyWidthGet(&sContext);
813             sRect.i16YMax = 64;
814             GrContextForegroundSet(&sContext, ClrBlack);
815             GrRectFill(&sContext, &sRect);
816             break;
817         case numeric:
818             sRect.i16XMin = 0;
819             sRect.i16YMin = 48;
820             sRect.i16XMax = GrContextDpyWidthGet(&sContext);
821             sRect.i16YMax = 64;
822             GrContextForegroundSet(&sContext, ClrBlack);
823             GrRectFill(&sContext, &sRect);
824             GrContextForegroundSet(&sContext, ClrWhite);
825             sprintf(tempStr,"%i",pui32ADC0Value[2]);
826             GrStringDrawCentered(&sContext, tempStr, -1,
```

```
827                     GrContextDpyWidthGet(&sContext) / 2, 56, 16);
828           break;
829         case histogram:
830           sRect.i16XMin = 0;
831           sRect.i16YMin = 48;
832           sRect.i16XMax = val[2];
833           sRect.i16YMax = 64;
834           GrContextForegroundSet(&sContext, ClrDarkBlue);
835           GrRectFill(&sContext, &sRect);
836           sRect.i16XMin = val[2];
837           sRect.i16XMax = 96;
838           GrContextForegroundSet(&sContext, ClrBlack);
839           GrRectFill(&sContext, &sRect);
840           break;
841       }
842     }                                           // End indefinite while()
843   }                                             // End of main()
844
845   //*****************************************************************************
846   //
847   // PuTTY window clearing function.
848   //
849   //*****************************************************************************
850   void clear() {
851     UARTCharPut(UART0_BASE, 12);
852   }
853
854   //*****************************************************************************
855   //
856   // Using the character output function as a base for a parent function
857   // used to output an entire string to the OLED one character at a time.
858   //
859   //*****************************************************************************
860   void putString(char *str) {
861     for(int i = 0; i < strlen(str); i++) {
862       UARTCharPut(UART0_BASE, str[i]);
863     }
864   }
865
866   //*****************************************************************************
867   //
868   // Print menu function that takes the complete menu as a string and
869   // utilises the print string function created above to output the entire menu
870   // in one transmission block to the PuTTY window.
871   //
872   //*****************************************************************************
873   void printMenu() {
874       // Menu below is a multiline string declaration ONLY FOR PRINTING FORMAT
875     char*menu = "\rMenu Selection: \n\rP - Party Mode\n\rC - Change Background
876           Color\n\rE - Erase Terminal Window\n\rL - Flash LED\n\rF - Flood
877           Character\n\rM - Print the Menu\n\r1- Toggle Display Mode for First
878           Potentiometer\n\r2- Toggle Display Mode for Second Potentiometer\n\r3-
879           Toggle Display Mode for Third Potentiometer\n\rQ - Quit this \
880           program\n\r";
881     putString(menu);
882   }
883
884   //*****************************************************************************
885   //
886   // Blinky LED "heartbeat" function.
887   //
888   //*****************************************************************************
889   void blinky(volatile uint32_t ui32Loop) {
890     // Turn on the LED.
891     GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_2, GPIO_PIN_2);
892
893     // Delay for amount specified in the LEDon #define at the top.
894     for(ui32Loop = 0; ui32Loop < LEDon; ui32Loop++) {
895     }
896
897     // Turn off the LED.
898     GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_2, 0);
899
900     // Delay for amount specified in the LEDoff #define at the top.
901     for(ui32Loop = 0; ui32Loop < LEDoff; ui32Loop++) {
902     }
903
904   }
```

APPENDIX B: TivaWare© Board Connections



The long bar is the potentiometer's sliding wiper.

This device is one of the three potentiometers used for this exercise.

The blue wire is the GND connection to the board and to all connections along the same horizontal row of ports (on the white board).

The pink wire is the power connection, or the hot wire, which supplies +3.3V to the board and all connections along the same horizontal row of ports (on the white board).

The yellow, blue, and red wires, specified above, connect the individual potentiometers to the TivaWare® TM4C123G Board using separate ports (4, 5, and 6) so their output to the OLED can be independently controlled.

APPENDIX C: Instructor Sign Off

Instructor Sign-off

Each laboratory group must complete a copy of this sheet.

Student A Name Andrea Gray

Student B Name Daniel Pietz

DK-TM4C123G Board Used 34

Lab Station Used 04

IAR EWARM Version Used 7.70.

TI Tivaware Version Used _____

1. Demonstrate operation of your program which displays a "splash" / credits screen to the OLED and displays a value read in from at least one selected analog channel {Ain4-Ain7 typical} as a decimal or hexidecimal number. Your program should continue to have menu-driven UART connectivity, but the display requirements are limited for this signoff.

(3) Bar Decimal Scaled 0-95
Party Mode

Instructors's Initials _____ Date/Time 30-JAN 4:08PM

2. Demonstrate operation of your program which allows the display of both numeric value and graphical display for at least (3) analog channels present on the DK-TM4C123G board. The other requirements of menu driven operation, LED and splash screen must also be met.

   Answer any questions the lab faculty may have, and demonstrate to the lab faculty that you have met the requirements of this laboratory exercise.

Awesome Splash - Bouncing Ball Animated
Blinky toggle

Instructors's Initials _____ Date/Time 30-JAN 4:10pm

6 /5

All Reqmts Met

Instructor Sign-off

Each laboratory group must complete a copy of this sheet.

Student A Name Andrea Gray

Student B Name Daniel Pietz

DK-TM4C123G Board Used 34

Lab Station Used 04

IAR EWARM Version Used 7.70.1

TI Tivaware Version Used 2.1.4.178

1. Demonstrate operation of your program which displays a "splash" / credits screen to the OLED and displays a value read in from at least one selected analog channel {Ain4-Ain7 typical} as a decimal or hexidecimal number. Your program should continue to have menu-driven UART connectivity, but the display requirements are limited for this signoff.

Bouncing Ball Splash!
Bonus Paddle mode ✓

Instructors's Initials _____ Date/Time 1255 30JAN19

2. Demonstrate operation of your program which allows the display of both numeric value and graphical display for at least (3) analog channels present on the DK-TM4C123G board. The other requirements of menu driven operation, LED and splash screen must also be met.

   Answer any questions the lab faculty may have, and demonstrate to the lab faculty that you have met the requirements of this laboratory exercise.

Instructors's Initials _____ Date/Time 30JAN19 1255

5 /5 +1 for Splash

APPENDIX D: Definitions

- **ADC—**Analog to Digital Converter
- **OLED**—Organic Light Emitting Diode on the TivaWare® TM4C123G Board
- **PDL**—Peripheral Driver Library
- **Potentiometer**—An instrument for measuring electric motor force by balancing it against the potential difference produced by passing a known current through a known variable resistance.[3]
- **Sliding Wiper**—On the potentiometer, this is the mechanical knob in which the user can control the analog input values. This works by moving along the resistive element in the potentiometer itself.
- **GND**—The ground pin on the TivaWare® TM4C123G Board.

APPENDIX E: Citation

[1] : *Tiva™ TM4C123GH6PM Microcontroller: Data Sheet.* Rev. E. [eBook]. Austin: Texas Instruments, 2014, p.898. Available at: http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf. [Accessed: Jan. 29, 2019].

[2] : *Using the Stellaris® Microcontroller Analog-to-Digital Converter (ADC): Application Note AN01247.* [eBook]. Austin: Texas Instruments, 2009, p. 6-11. Available at: http://www.ti.com/litv/pdf/spma028. [Accessed: Jan. 29, 2019].

[3] : Oxford Unviersity Press, "Oxford American College Dictionary: Potentiometer," 2002. [Online]. Available at: https://www.google.com/search?q=potentiometer+definition&rlz=1C5CHFA_enUS751US751&oq=potentiometer+de&aqs=chrome.1.69i57j35i39j0l4.6272j1j7&sourceid=chrome&ie=UTF-8. [Accessed: Feb. 10, 2019].

[4] : Plutarch. [Quote]. 50 A.D. – 120 A.D.