	EMBRY-RIDDLE AERONAUTICAL UNIVERSITY **Department of Electrical, Computer, and Software Engineering** **CEC322**: Microprocessor Laboratory (Spring 2019), **SECTION** PC51 Due Date: 29-JAN-2019 Turned in: 29-JAN-2019		
	PREPARED BY:	**EQUIPMENT S/N:**	**PERFORMED:**
	Gray, Andrea Pietz, Daniel	DKTM4C123G	22-JAN-2019

LABORATORY #2

Use of UART/Serial Communication to/from the TivaWare® DK-TM4C123G Board

OBJECTIVES

- Increase the usage and knowledge of the DK-TM4C123G Development Kit
- Implement UART communication between DK-TM4C123G Board and the PC workstation
- Differentiate between UART button press events
- Establish control of the DK-TM4C123G Board functionality through a unique, reliable menu interface
- Continued use of 'C' programming language
- Increased understanding and use of functions, variables, and constants in the TivaWare® Peripheral Driver Library

PURPOSE

To establish a connection through a virtual COM port and allow a user to control multiple devices, such as the OLED display and LED out, through a displayed menu screen on the DK-TM4C123G Development Kit. The OLED should display software maintained and continuously updated numerical values which can be useful in later debugging.

SPECIFICATIONS

The objective of the code development in this procedure is to write and execute a program which continues the pseudo-infinite loop based on the previous laboratory exercise. The program will utilize the OLED and LED functions to display the satisfactory UART connection and communication.

The program *must* implement the use of a menu function with two unique and properly functioning additional menu options. A constant communication must be maintained with the PC via UART ensuring a steadfast display of LED operation, loop/button counts, and I/O counts. Toggle functions must also be incorporated for the LED and "flood" character.

At minimum, the program is required to display a unique "splash-screen", a bi-directional UART communication, and the specifications described in short above.

PROCEDURE

Part 1:

The procedure's onset is easier to set up from the gained knowledge of the use of the IAR workbench from the previous laboratory exercise. To begin the IAR workbench should be set up in the same general manner as CS322 exercise 1. The base files to implement and work off for this procedure are the uart_echo file in conjunction with the completed file from exercise 1.

After the initial retrieval, organization, and verification process has been established and the communication settings have been specified between the PC and the DK-TM4C123G UART peripheral the PuTTY application should be opened and waiting for instruction from the IAR workbench.

The set up for this lab, while familiar, demonstrated the delicate nature of the IAR workbench settings in connection with the DK-TM4C123G Development Kit. It is imperative to understand the hardware and software aspects that will be utilized throughout the exercise to ensure a productive and timely procedural execution.

In laboratory exercise 2 a simple insight into the UART's communication set up and execution will allow for the focus of the lab to be maintained on the code execution and correction. The UART communication is present through a USB connection and the microcontroller pins U0RX and U0TX. This information may seem irrelevant in the inauguration of the process but becomes an indispensable bit of information in the later section of the exercise procedure.

To begin the construction of the project the cardinal objective, and arguably the most effortless task is to change the uart-echo string that displays as the "splash-screen" to a unique, team-specific message. The string "Gray & Pietz" was chosen for this program.

Subsequently, the code to enable the UART0 operations is implemented. This code segment employs the usage of interrupts. While not disadvantageous, the maintenance of the interrupt functionality is frivolous when the other required portions of the code obligate attention and adjustment. Additionally, the ROM_ and MAP_ prefixes are unnecessary, and their removal is recommended.

After the initial revision and simplification of the code for the program is completed the incorporation of the polled UART I/O functionality is invoked. This portion, with the implementation of the user defined process_menu() function and UART checking/gathering functions, seems daunting and the function and constant names are unfamiliar. However, the introductory exposure to the UART communication system code is (fairly) straight forward and does not require multiple application attempts to compile fully. This is a nice, but fleeting, surprise.

For the first process_menu() subfunction, the employment of a unique "flood" character is transmitted once per second. Since the prefatory introduction to the DK-TM4C123G Development Kit does not include inessential features such as the use of the built-in crystal main internal clock, the process of ensuring the "flood" character's ('@' in this instance) reiteration per second proves to be the first considerable challenge faced in this exercise. In the end, the timing was speculated, and the "flood" character was output after a certain number of loop iterations in the main process.

The following step is to create the full layout of the menu_process() function to the OLED display. The menu functionalities are as shown below in Figure 1.0.

```
Menu Selection:
P - Party Mode
C - Change Background Color
E - Erase Terminal Window
L - Flash LED
F - Flood Character
M - Print the Menu
Q - Quit this program
```

Figure 1.0 : Menu Functionalities

As shown above, the program offers the options to change the color of the OLED screen banner, clear the output shown in the PuTTY window, toggle the “flood” character output, toggle the LED function, a party mode toggle which loops through different OLED color displays, reprinting the menu display, and the option to quit the program (with a goodbye message output to the OLED and PuTTY window). Out of all of these options the hardest to materialize is the clear function due to the fact that the PuTTY window will show all formatting errors (and lack of formatting). The realization of the considerable differences between the DK-TMC123G board programming and the routine programming language (with the common built-in syntax and formatting) is expeditiously overt in the literal manifestation of the program.

Part 2:

The use of the DK-TM4C123G Development Kit’s output signals has primarily been visualized through COM communication and OLED interface interpretation. To show the output from the UART0_RX and UART0_TX pins, the board is connected to the Analog Discovery monitor and projected on the Waveforms graphical application.

The figure offered with the laboratory exercise 2 instruction manual for the demonstration of the Analog Discovery connections to the two pins mentioned above on the DK-TM4C123G board is a critical figure in the establishment of communication between the board and the software. Unfortunately, the shadowed figure offered, reproduced below in Figure 1.1, is quite difficult to decipher and a substantial amount of time is liable to succumb to its essential execution which is detrimental to the performance of the Waveforms program if misinterpreted (witnessed from experience).

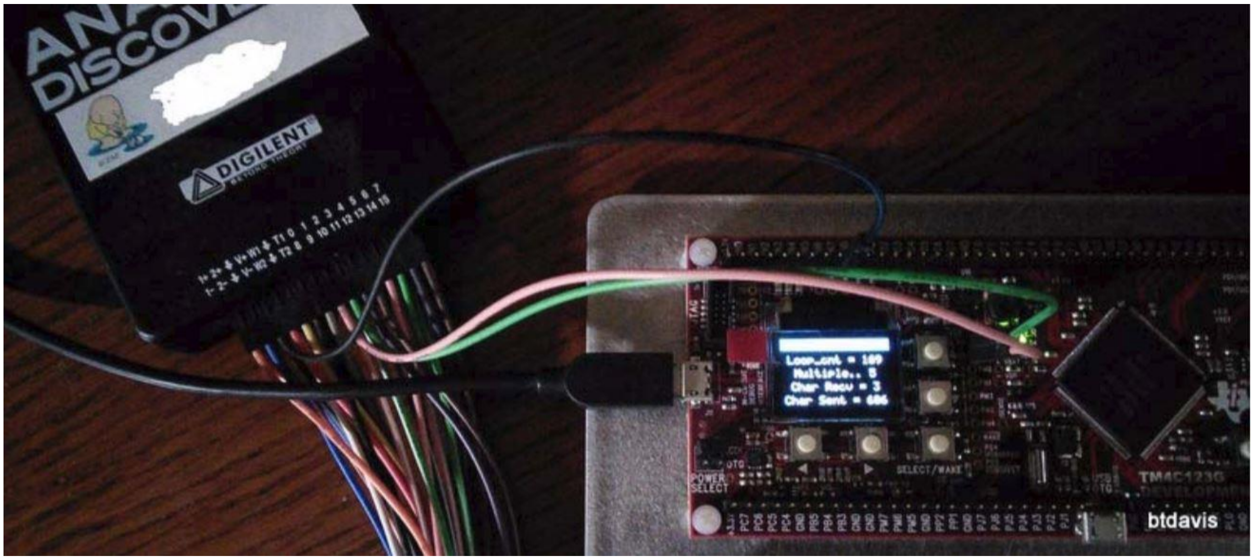


Figure 1.1: Connection Confusion

Once the correct pin/port locations are determined, the consequent trial proved to be the undertaking of the Waveforms program. This application does not reflect the image offered in the laboratory exercise manual and a tutorial to the Waveforms software is not offered. This miscommunication can lead to a considerable misapplication of time as it proved for us.

Once the software was functioning according to the exercise's constraints the capture of the waveforms of the "flood" character output and an extended sequence of characters, shown below in Appendix B, was uncomplicated. The waveforms show the ASCII output interpretation and UART interpreters and an additional display of the harmonious functionality between the hardware and our software development – a rewarding and invigorating sight for sore eyes.

REPORT DISCUSSION

1. PDL functions used for the first time by our team in this lab:
   i. UARTPutChar( )
   ii. GrContextBackgroundSet( )
   iii. UARTCharsAvail( )
   iv. GrRectFill( )
2. The key difference between blocking and non-blocking I/O is that with the former the thread is unable to proceed until the input is processed fully. Moreover, with a blocking I/O there can only be one input at a time when antagonistically a non-blocking I/O can take multiple threads. The trade-off when using the non-blocking I/O is that there is less of a guarantee that the data transmitted was assuredly received.
3. The ASCII value for the "flood" character used in this program, depicted in Appendix B: Figure 1, is 40. The hexadecimal value 40, or h40 as shown in the UART Waveform interpreter, represents the '@' symbol.
4. The ASCII value for the character shown ('T') being transmitted by the DK-TM4C123G in Figure 5 in the CS322 Laboratory Exercise 2 Procedure Manual is 54.
5. After examination of Figure 6 from the CS322 Laboratory Exercise 2 Procedure Manual, it is clear to conclude that the shown signal {DIO0} corresponds to UART0_TX and the signal {DIO1} corresponds to UART0_RX. This is apparent considering that the Waveform graph is depicting the

input of a character to the PuTTY menu which then outputs the appropriate response to the menu call to the DK-TM4C123G. Therefore, it is safe to conclude that the output (TX/Transmission) corresponds to the {DIO0} signal due to the significant total of waves being produced in the figure comparative to the single waveform being shown for the {DIO1} signal (a single input character).

6. UART communication is implemented in the TivaWare© DK-TM4C123G by sending a sequence of 8-bit messages in parallel. At the output of the UART signal, these bits are usually represented by logical voltage levels. The RS-232 specifies voltage levels and larger voltage fluctuations make the RS-232 more resistant to interference. A microcontroller UART cannot generate ±15 voltages by itself. A RS-232 connection is much safer in voltage connections without damage because of the more universal standards and less need for converters and calculations. The voltage of the {idle signal (1), start-bit (0), stop-bit (1)} in UART and RS-232 is {+3.3, 0, +3.3} and {+15, (+3 to +15), (-3 to -15)} respectively. [1] The board used has a voltage input of +3.3 V so that is the data that is used above. In the TivaWare© manual though, it states: "for both low-power and normal mode operation, the ILPDVSR field in the UARTILPR register must be programmed such that 1.42 MHz < FIrLPBaud16 < 2.12 MHz" [2]; so, the true value could be interpreted as such.
7. The Cytron URS232A converter is an industry standard integrated circuit Driver / Receiver which can be used to convert from TTL-Level UART to an RS-232 level UART.
8. The most unanticipated part of the lab was the difficulty of configuring the Waveforms software and the difficulty of working with a substantial amount of code, with all its bugs, in such a limited time frame. Although the implementation of the specific baseline requirements in the laboratory exercise were not strenuous, the incorporation of each individual part into the larger program proved to be formidable. The laboratory exercise timing calculation does not consider how many diminutive errors can be produced in a single block of code; and furthermore, how ambiguous they will seem. After all is resolved and concluded it is safe to assert that a greater effort in research into supplemental software applications would be much more helpful on the student end in the future and an increased photo resolution and varied camera angles for figures would make this exercise go more fluid in the future.

CONCLUSION

The trials, the errors, the frustrating compiling errors, and the relieving breakthroughs all intermingle into a programming experience unparalleled by the novice encounters that have preceded this exercise. The lessons learned throughout the process reflect themes of time management, extended research and preparation efforts, and the realization that the absence of one semi-colon could mean hours of frustrating contemplation. Despite the foreseeable triumphs and tribulations there remains a steadfast ray of awe and admiration at the recognition that the threshold between hardware and software lies a keystroke away.

APPENDICES:

APPENDIX A: Lab Code

```
//*****************************************************************************
//
// CS322.50 Labratory 2 Software File
// Developed by: Andrea Gray and Daniel Pietz (c)
// Version: 1.30 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.
//
//*****************************************************************************

//*****************************************************************************
//
//                        !!! DISCLAIMER !!!
//
// Functions in the code below have been wrapped without syntax consideration
// in referfence to compiling to fit lab report width specifications.
// The code file (.c) used in the execution of the project does not include
// these text wraps for proper compilation.
//
//*****************************************************************************

#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 <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/fpu.h"              // Header for FPULazyStackingEnable
                                        // function
#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

//*****************************************************************************
//
// This example application utilizes the UART to echo text. All characters
// recoeved on the UART are transmitted back to the UART.
//
//*****************************************************************************


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

//*****************************************************************************
//
// The error routine that is called if the driver library encounters an error.
//
```

```c
//*****************************************************************************
#ifdef DEBUG
void __error__(char *pcFilename, uint32_t ui32Line) {
}
#endif

//*****************************************************************************
//
// Holds the current, debounced state of each button.  0 = pressed.
// We assume that we start with all the buttons released (though if one is
// pressed when the application starts, this will be detected).
//
//*****************************************************************************
static uint8_t g_ui8ButtonStates = ALL_BUTTONS;

//*****************************************************************************
//
// Initializes the GPIO pins used by the board pushbuttons with a weak
// pull-up.
//
//*****************************************************************************
void ButtonsInit(void) {
    //
    // Enable the GPIO port to which the pushbuttons are connected.
    //
    SysCtlPeripheralEnable(BUTTONS_GPIO_PERIPH);

    //
    // Set each of the button GPIO pins as an input with a pull-up.
    //
    GPIODirModeSet(BUTTONS_GPIO_BASE, ALL_BUTTONS, GPIO_DIR_MODE_IN);
    GPIOPadConfigSet(BUTTONS_GPIO_BASE, ALL_BUTTONS,
                     GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);

    //
    // Initialize the debounced button state with the current state read from
    // the GPIO bank.
    //
    g_ui8ButtonStates = GPIOPinRead(BUTTONS_GPIO_BASE, ALL_BUTTONS);
}

//*****************************************************************************
//
// Send a string to the UART.
//
//*****************************************************************************
void UARTSend(const uint8_t *pui8Buffer, uint32_t ui32Count) {
    // Loop while there are more characters to send.
    while(ui32Count--) {
        // Write the next character to the UART.
        UARTCharPutNonBlocking(UART0_BASE, *pui8Buffer++);
    }
}


//*****************************************************************************
//
// The main function to intialize the UART, LED, OLED, and run through the
// program
//
//*****************************************************************************
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

    //*************************************************************************
    // Counter variables
    //*************************************************************************
    int tickCount = 0;                                  // Clock ticks for flood
                                                        // character.
```

```c
    int whileLoop = 1;
    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 OLED
                                                            // output.

    //*****************************************************************************
    //
    // Enable lazy stacking for interrupt handlers.  This allows floating-point
    // instructions to be used within interrupt handlers, but at the expense of
    // extra stack usage.
    //
    //*****************************************************************************
    FPULazyStackingEnable();

    //*****************************************************************************
    //
    // Set the clocking to run directly from the crystal.
    //
    //*****************************************************************************
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
                   SYSCTL_XTAL_16MHZ);

    //*****************************************************************************
    //
    // Enable the GPIO port that is used for the on-board LED.
    //
    //*****************************************************************************
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);

    //*****************************************************************************
    //
    // Check if the LED peripheral access is enabled.
    //
    //*****************************************************************************
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOG)) {
    }

    //*****************************************************************************
    //
    // Enable the GPIO pin for the LED.  Set the direction as output, and
    // enable the GPIO pin for digital function.
    //
    //*****************************************************************************
    GPIOPinTypeGPIOOutput(GPIO_PORTG_BASE, GPIO_PIN_2);


    //*****************************************************************************
    //
    // Initialize the OLED display driver.
    //
    //*****************************************************************************
    CFAL96x64x16Init();

    //*****************************************************************************
    //
    // Initialize the OLED graphics context.
    //
    //*****************************************************************************
    GrContextInit(&sContext, &g_sCFAL96x64x16);

    //*****************************************************************************
    //
    // Fill the top part of the screen with blue to create the banner.
    //
    //*****************************************************************************
    sRect.i16XMin = 0;                                      // left edge
    sRect.i16YMin = 0;                                      // top edge
    sRect.i16XMax = GrContextDpyWidthGet(&sContext) - 1;// right edge
    sRect.i16YMax = 9;                                      // bottom edge
    GrContextForegroundSet(&sContext, ClrDarkBlue);         // dark blue banner
```

```
    GrRectFill(&sContext, &sRect);                          // fill the rectangle
                                                            // area

    //*****************************************************************************
    //
    // 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);

    //*****************************************************************************
    //
    // Initialize the display and write instructions.
    //
    //*****************************************************************************
    GrStringDrawCentered(&sContext, "Choose an", -1,
                         GrContextDpyWidthGet(&sContext) / 2, 20, false);
    GrStringDrawCentered(&sContext, "option", -1,
                         GrContextDpyWidthGet(&sContext) / 2, 30, false);
    GrStringDrawCentered(&sContext, "from the", -1,
                         GrContextDpyWidthGet(&sContext) / 2, 40, false);
    GrStringDrawCentered(&sContext, "menu.", -1,
                         GrContextDpyWidthGet(&sContext) / 2, 50, false);

    //*****************************************************************************
    //
    // Enable the peripherals used by this example.
    //
    //*****************************************************************************
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);        // GPIO Pin Set
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);        // UART0 Set


    //*****************************************************************************
    //
    // Set GPIO 0 and 1 as UART pins.
    //
    //*****************************************************************************
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //*****************************************************************************
    //
    // Configure the UART for custom 115,200 baud rate.
    //
    //*****************************************************************************
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                         UART_CONFIG_PAR_NONE));


    //*****************************************************************************
    //
    // Check to see if the GPIO peripheral is ready.
    //
    //*****************************************************************************
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOG)) {
    }

    //*****************************************************************************
    //
    // Set the LED Pin output to pin 2.
    //
    //*****************************************************************************
    GPIOPinTypeGPIOOutput(GPIO_PORTG_BASE, GPIO_PIN_2); // GPIO output is pin 2.
    ButtonsInit();                                          // Enable buttons for
                                                            // each loop iteration.
    int Looper = 0;                                         // Set loop variable and
                                                            // initialize to 0.
```

```
    clear();
    printMenu();                                               // Print the menu out.

    //*****************************************************************************
    //
    // Indefinite while loop
    //
    //*****************************************************************************
    while(whileLoop != 0) {

        // Variable incrementations.
        Looper++;
        whileLoop++;
        tickCount++;

        //*************************************************************************
        //
        // Flood character output description and timing.
        //
        //*************************************************************************
        if(tickCount == 3) {
            tickCount = 0;
            if(shouldFlood == 1) {
                UARTCharPut(UART0_BASE, '@');                  // Toggle catch
            }
        }

        //*************************************************************************
        //
        // Blinky function toggle catch
        //
        //*************************************************************************
        if(shouldBlink) {
            blinky(ui32Loop);
        }

        //*************************************************************************
        //
        // Checking for a button press and displaying output as designated and
        // incrementing counts accordingly.
        //
        //*************************************************************************
        buttonState = (~GPIOPinRead(BUTTONS_GPIO_BASE,ALL_BUTTONS)
                        && ALL_BUTTONS);
        if(GPIOPinRead(BUTTONS_GPIO_BASE, ALL_BUTTONS)) {
            buttonCounter++;
            switch (buttonState) {
                case UP_BUTTON:
                    lastPressed = 1;                           // UP = 1
                    break;
                case DOWN_BUTTON:
                    lastPressed = 2;                           // DOWN = 2
                    break;
                case LEFT_BUTTON:
                    lastPressed = 3;                           // LEFT = 3
                    break;
                case RIGHT_BUTTON:
                    lastPressed = 4;                           // RIGHT = 4
                    break;
                case SELECT_BUTTON:
                    lastPressed = 5;                           // SELECT = 5
                    break;
            }
        }

        //*************************************************************************
        //
        // Changing the OLED output depending on Party Mode toggle and Banner
        // Color menu selection.
        //
        //*************************************************************************
        if(shouldCycle == 1) {
            if(colorSwitch == 2) {
                colorSwitch = -1;                              // End case catch
            }

            // Setting OLED display as full avaliable rectangle output.
            colorSwitch++;
```

```
        sRect.i16XMin = 0;
        sRect.i16YMin = 0;
        sRect.i16XMax = GrContextDpyWidthGet(&sContext) - 1;
        sRect.i16YMax = GrContextDpyHeightGet(&sContext) - 1;

        // Switch case to determine the next OLED color output.
        switch (colorSwitch) {
            case 0:                                 // 0 = Dark Blue
                GrContextForegroundSet(&sContext, ClrDarkBlue);
                GrContextBackgroundSet(&sContext, ClrDarkBlue);
                break;
            case 1:                                 // 1 = Red
                GrContextForegroundSet(&sContext, ClrRed);
                GrContextBackgroundSet(&sContext, ClrRed);
                break;
            case 2:                                 // 2 = Green
                GrContextForegroundSet(&sContext, ClrGreen);
                GrContextBackgroundSet(&sContext, ClrGreen);
                break;
            default:                                // Default = Dark Blue
                GrContextForegroundSet(&sContext, ClrDarkBlue);
                GrContextBackgroundSet(&sContext, ClrDarkBlue);
                break;
        }

        // Re-draw the OLED rectangular output with above determined color
        // and correct banner message.
        GrRectFill(&sContext, &sRect);
        GrContextForegroundSet(&sContext, ClrWhite);
        GrContextFontSet(&sContext, g_psFontFixed6x8);
        GrStringDrawCentered(&sContext, "Gray & Pietz", -1,
                             GrContextDpyWidthGet(&sContext) / 2, 4, 0);
    }                                                          // end shouldCycle()

    //*************************************************************************
    //
    // While loop when a character is inputted in the PuTTY window.
    //
    //*************************************************************************
    while(UARTCharsAvail(UART0_BASE)) {
        char str[50];                               // Empty string
                                                    // declaration for
                                                    // line 469
        int32_t local_char;                         // Setting the input
                                                    // character to a
                                                    // local variable.

        //*********************************************************************
        //
        // Re-draw the OLED with updated statistics
        //
        //*********************************************************************
        sRect.i16XMin = 0;
        sRect.i16YMin = 10;
        sRect.i16XMax = GrContextDpyWidthGet(&sContext) - 1;
        sRect.i16YMax = GrContextDpyHeightGet(&sContext) - 1;
        GrContextForegroundSet(&sContext, ClrBlack);
        GrRectFill(&sContext, &sRect);
        GrContextForegroundSet(&sContext, ClrWhite);
        sprintf(str,"Loop: %d",Looper);
        GrStringDrawCentered(&sContext, str, -1,
                             GrContextDpyWidthGet(&sContext) / 2, 20, false);
        sprintf(str,"Last Pressed: %d",lastPressed);
        GrStringDrawCentered(&sContext, str, -1,
                             GrContextDpyWidthGet(&sContext) / 2, 30, false);
        sprintf(str,"Press Counter: %d",buttonCounter);
        GrStringDrawCentered(&sContext, str, -1,
                             GrContextDpyWidthGet(&sContext) / 2, 40, false);
        local_char = UARTCharGetNonBlocking(UART0_BASE);

        //*********************************************************************
        //
        // If the input character is not invalid, begin the character
        // matching statment below through the switch statment and act
        // accordingly to the user input.
        //
        //*********************************************************************
        if (local_char != -1) {
```

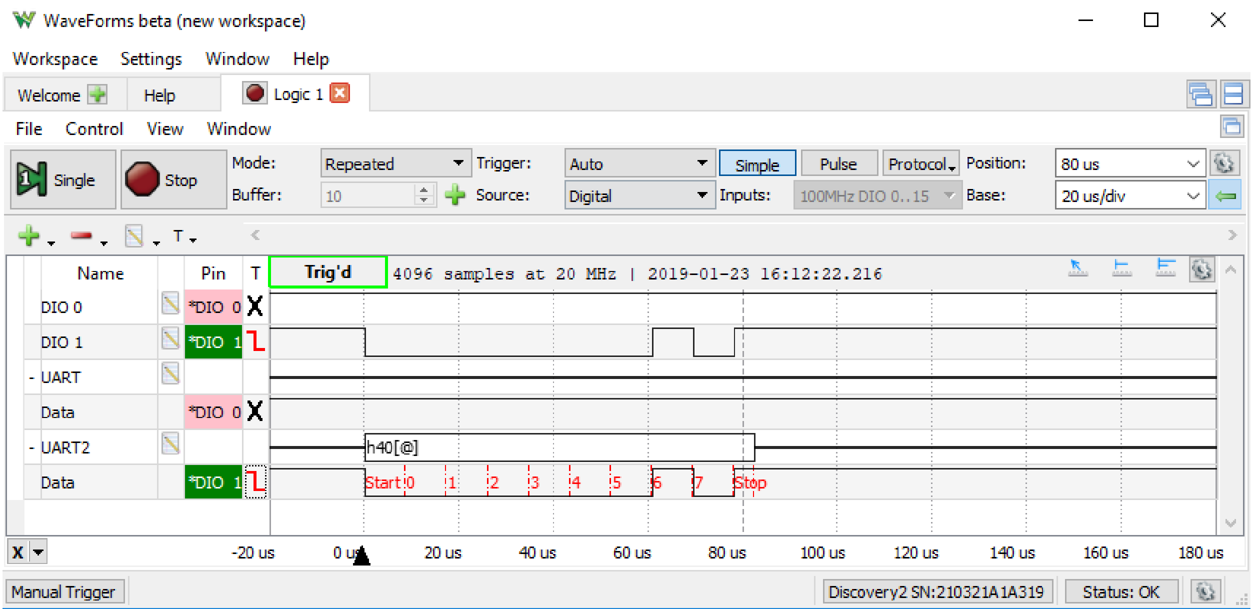
```c
            gNumCharRecv++;                         // Characters recieved
                                                    // counter incrementation
            UARTCharPut(UART0_BASE, local_char);    // Sending a single
                                                    // character through the
                                                    // UART TX (transmitting)
                                                    // channel

            // Begin character input matching to menu option.
            switch(local_char) {
                case 67:                            // Switch banner color
                                                    // - C
                    if(colorSwitch == 2) {
                        colorSwitch = -1;
                    }
                    colorSwitch++;                  // Color cycle counter
                                                    // incrementation

                    // sRect below is specifying banner area.
                    sRect.i16XMin = 0;
                    sRect.i16YMin = 0;
                    sRect.i16XMax = GrContextDpyWidthGet(&sContext) - 1;
                    sRect.i16YMax = 9;
                    switch (colorSwitch) {
                        case 0:
                            GrContextForegroundSet(&sContext, ClrDarkBlue);
                            GrContextBackgroundSet(&sContext, ClrDarkBlue);
                            break;
                        case 1:
                            GrContextForegroundSet(&sContext, ClrRed);
                            GrContextBackgroundSet(&sContext, ClrRed);
                            break;
                        case 2:
                            GrContextForegroundSet(&sContext, ClrGreen);
                            GrContextBackgroundSet(&sContext, ClrGreen);
                            break;
                        default:
                            GrContextForegroundSet(&sContext, ClrDarkBlue);
                            GrContextBackgroundSet(&sContext, ClrDarkBlue);
                            break;
                    }

                    // Filling in the rest of the OLED screen after banner is
                    // updated.
                    GrRectFill(&sContext, &sRect);
                    GrContextForegroundSet(&sContext, ClrWhite);
                    GrContextFontSet(&sContext, g_psFontFixed6x8);
                    GrStringDrawCentered(&sContext, "Gray & Pietz", -1,
                                         GrContextDpyWidthGet(&sContext) / 2,
                                         4, 0);
                    break;

                case 69:                            //Clear PuTTY window - E
                    clear();
                    break;

                case 70:                            // Flood toggle - F
                    if(shouldFlood == 0) {
                        shouldFlood = 1;
                    }
                    else {
                        shouldFlood = 0;
                    }
                    break;

                case 76:                            //LED toggle - L
                    if(shouldBlink == 0)
                        shouldBlink = 1;
                    else
                        shouldBlink = 0;
                    break;

                case 77:                            // Re-print menu - M
                    printMenu();
                    break;

                case 80:                            // Party Mode Toggle - P
                    if(shouldCycle == 0) {
                        shouldCycle = 1;
```

```
                        }
                        else
                        {
                            shouldCycle = 0;
                        }
                        break;

                    case 81:                                // Quit program - Q
                        putString("\n\rBYE!");              // Goodbye message to
                                                            // PuTTY window.

                        // Re-draw OLED with goodbye statment in red font.
                        sRect.i16XMin = 0;
                        sRect.i16YMin = 0;
                        sRect.i16XMax = GrContextDpyWidthGet(&sContext) - 1;
                        sRect.i16YMax = GrContextDpyHeightGet(&sContext) - 1;
                        GrContextForegroundSet(&sContext, ClrBlack);
                        GrContextBackgroundSet(&sContext, ClrBlack);
                        GrRectFill(&sContext, &sRect);
                        GrContextForegroundSet(&sContext, ClrRed);
                        GrContextFontSet(&sContext, g_psFontFixed6x8);
                        GrStringDrawCentered(&sContext, "Goodbye", -1,
                                             GrContextDpyWidthGet(&sContext) / 2,
                                             30, false);
                        whileLoop = 0;
                        break;

                }                                           // End menu switch block
            }                                               // End valid character
                                                            // check
        }                                                   // End character
                                                            // detection
                                                            // loop.
    }                                                       // End indefinite while()
}                                                           // End of main()

//*****************************************************************************
//
// PuTTY window clearing function.
//
//*****************************************************************************
void clear() {
    UARTCharPut(UART0_BASE, 12);
}

//*****************************************************************************
//
// Using the character output function as a base for a parent function
// used to output an entire string to the OLED one character at a time.
//
//*****************************************************************************
void putString(char *str) {
    for(int i = 0; i < strlen(str); i++) {
        UARTCharPut(UART0_BASE, str[i]);
    }
}

//*****************************************************************************
//
// Print menu function that takes the complete menu as a string and
// utilizes the print string function created above to output the entire menu
// in one transmission block to the PuTTY window.
//
//*****************************************************************************
void printMenu() {
    // string below is wrapped for printing and elongated for compiling.
    char*menu = "\rMenu Selection: \n\rP - Party Mode\n\rC - Change Background
                Color\n\rE - Erase Terminal Window\n\rL - Flash LED\n\rF
                - Flood Character\n\rM - Print the Menu\n\rQ - Quit this
                program\n\r";
    putString(menu, UART0_BASE);
}

//*****************************************************************************
//
// Blinky LED "heartbeat" function.
//
//*****************************************************************************
```

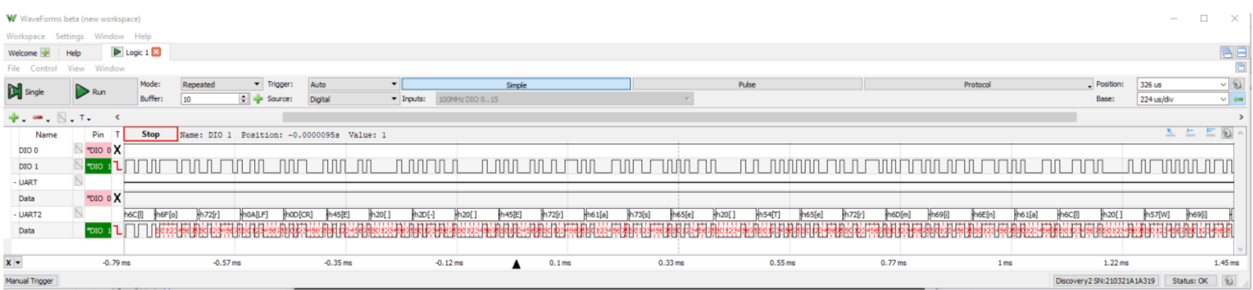
```c
void blinky(volatile uint32_t ui32Loop) {
    // Turn on the LED.
    GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_2, GPIO_PIN_2);

    // Delay for amount specified in the LEDon #define at the top.
    for(ui32Loop = 0; ui32Loop < LEDon; ui32Loop++) {
    }

    // Turn off the LED.
    GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_2, 0);

    // Delay for amount specified in the LEDoff #define at the top.
    for(ui32Loop = 0; ui32Loop < LEDoff; ui32Loop++) {
    }
}

//*******************************************************************************
// End of program!
//*******************************************************************************
```

APPENDIX B: Waveform Graphs

In the waveform graphs below, the UART0 input (RX-receiving) and output (TX-transmitting) signal displays are graphically exhibited. In Figure 1 the single "flood" character is shown with the ASCII interpreter below. In Figure 2 the menu_process() function output is delineated through its electrical transmittance.



Appendix B: Figure 1: Waveform



Appendix B: Figure 2: Extended Waveform

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 28

Lab Station Used 05

IAR EWARM Version Used 7.70.1

TI Tivaware Version Used 2.1.4.178

1. Demonstrate communication between a terminal program on the Windows workstation and the DK-TM4C123 development kit. Software should display menu and responses to menu selections to the UART and use these interactions to modify data being displayed to the OLED. **List** the specific functionality toggles implemented in the solution below.

   - "Flood" character toggle '@'
   - LED 'L'
   - Party Mode ✓

   Loop count
   Buttons All OP
   Great UI - Above Reqmts

   Instructors's Initials 23-Jan BA Date/Time 23-JAN 4:07PM

2. Verify the figure captured on the Analog Discovery is a valid UART character capable of being converted to ASCII by analysis of the signal.

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

   Instructors's Initials BA Date/Time 23-JAN 4:10PM 5 /5

January 17, 2019 10

APPENDIX D: Definitions

- **UART:** Universal Asynchronous Receiver/Transmitter
- **I/O:** input and output
- **OLED:** Organic LED display
- **DK-TM4C123G:** The Texas-Instrument development kit board utilized for the physical laboratory exercise implementation

APPENDIX E: Citation

[1] : Alexeev, Nick. (2014). *"Difference Between UART and RS-232."* [Online]. Available at: https://electronics.stackexchange.com/questions/110478/difference-between-uart-and-rs-232. [Accessed: 28 JAN 2019].

[2] : http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf. (2014). [eBook]. Austin: Texas Instruments, p.898. Available at: http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf. [Accessed: 29 JAN 2019].