	EMBRY-RIDDLE AERONAUTICAL UNIVERSITY **Department of Electrical, Computer, and Software Engineering** **CEC322**: Microprocessor Laboratory (Spring 2019), **SECTION** PC51 Due Date: 19-FEB-2019 Turned in: 27-FEB-2019		
	PREPARED BY:	**EQUIPMENT S/N:**	**PERFORMED:**
	Gray, Andrea	DK-TM4C123G	5-FEB-2019

LABORATORY #4

Introduction to Interrupt I/O

OBJECTIVES

- Increase knowledge and experience with the DK-TM4C123G Development Kit
- Introduce Interrupt and timer peripheral concepts and implications
- Utilize timer and interrupt functions and constants in TivaWare© for the first time
- Continued use of the 'C' programming language and ADC

PURPOSE

The purpose of this laboratory exercise is to strengthen the use and knowledge of the analog to digital conversion while introducing the implementation of timer interrupts in the existing code. The value read in by the analog to digital converter, or ADC, is used to control the interrupts. It is necessary to use two (2) timers with individually processed frequencies. The foreground program blinky, the UART, and the OLED, which are implemented in all the laboratory assignments prior, is used to ensure the programming is running, to switch between display modes, and to display the program data respectively.

SPECIFICATIONS

To have a successful software project for this laboratory exercise a faculty member must sign off (Appendix C) after each of the requirements are met. The program, running on the DK-TM4C123G board under the debug mode, will meet the requirements if it is configured to utilize two (2) timer generated interrupts controlled using the analog input. The project must also be able to service interrupts using the DK-TM4C123G board and its service routines. The timers must be displayed showing one at a constant 1 Hz frequency and one at a varying frequency from 1 Hz to a user determined maximum frequency which is controlled using the potentiometer-controlled ADC.

PROCEDURE

For this procedure, the finalized software project from laboratory exercise three is used as the base/source code with additions from the timer example project, timers.c (incorporated into this project using the #include statement shown in the Report Discussion 1.f). Since the timer example project is not used as the base file for this procedure, it was necessary to change the interrupt vector table contained in startup_ewarm.c.

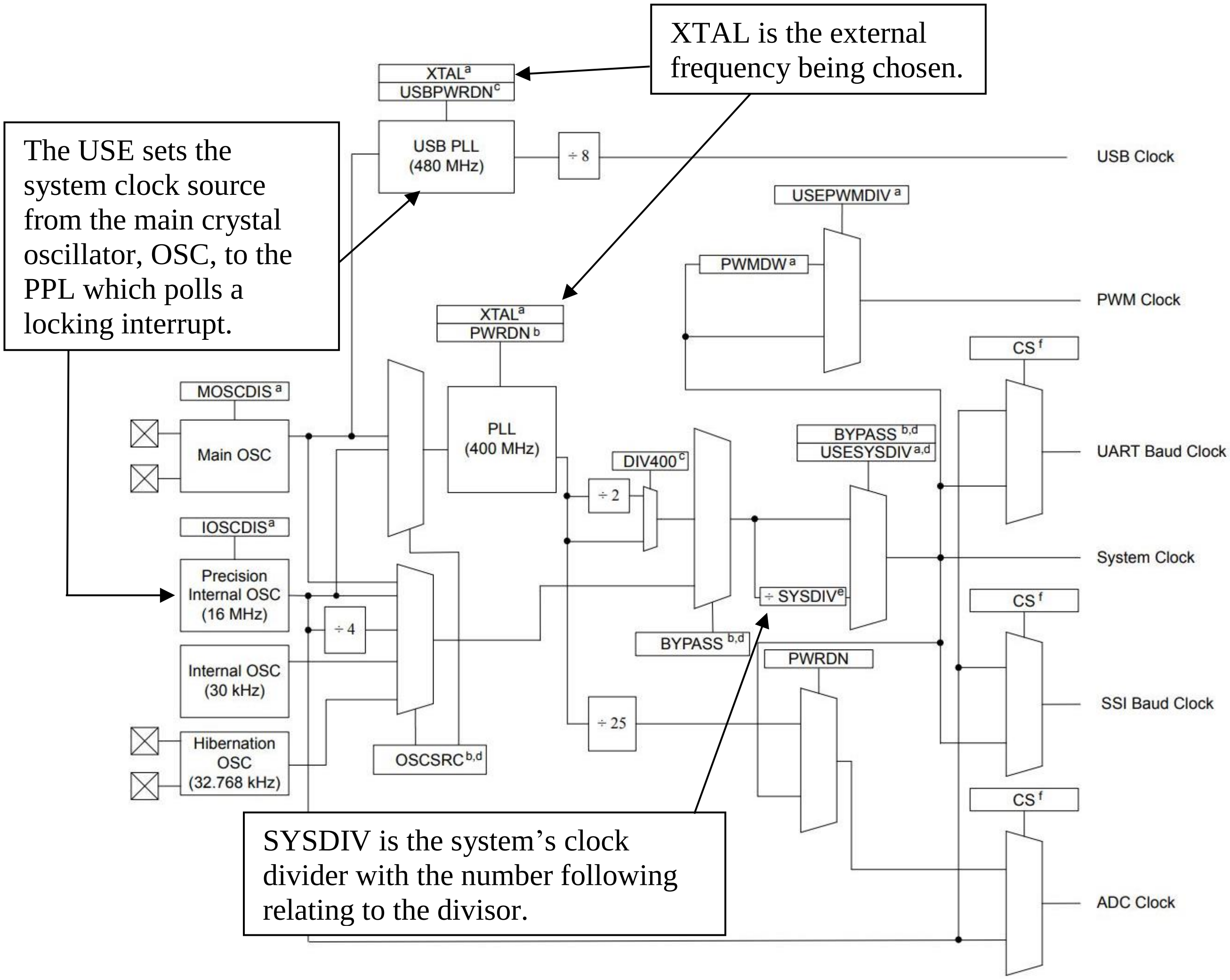
Process:

The process for this procedure is described below in the general timeline shown by the numbering below. The code retrieval is not included as a main step as it is described above and implied that it is already set up before the process for this exercise begins. Also, the splash screen for this procedure is maintained from the previous laboratory exercise and was not altered except for the title which depicts the change in team structure and the display of the SysCtlClockGet() function.

1. For this program, the code from the previous lab was cleaned up and sectioned out into separate functions to clear and organize the main function. The main function now only holds variable initializations, OLED and Clock initialization function calls, interrupt enable and disable functions, timer and peripheral set up functions, and the main infinite while loop. The infinite while loop, with interrupts enabled before the loop itself, consistently checks whether foreground programs like blinky and the flood character display should be enabled or not.
2. The main function's reorganization made is so many new global variables were created. These global variables were used because a lot of the variables are constantly being passed between functions.
3. Once the reorganization and set up of the software program for this exercise was complete, the next step was to configure the different operating frequency in SysCtlClockSet() with 16 MHz and 66 MHz and to initialize each of the peripherals. In this exercise the peripherals for the timers had to be set up along with the UART, ADC, and GPIO peripherals. These were set up in the main function and since the splash screen was not altered the next step was to modify the blinky call so that it was not affected by the interrupts.
4. Ensuring that the blinky call was not affected by the timer interrupts was not very difficult, as the only thing that needed to be done was ensure that the timer interrupt handler consistently checks for a UART character input if the user wants to turn the blinky on or off. Otherwise blinky will remain in the state that it is set to and continue to blink along with the passing of each infinite while loop iteration. The infinite while loop must be called continuously and therefore the blinky must maintain a steady 'heartbeat' to ensure visually that the program did not get stuck in an ISR.
5. While the timer interrupts were outlined by the timers.c file used, there were some trivial modifications that were needed to be done to ensure that this laboratory exercise executed properly. The interrupt service routine needs to display the SRV results, test the UART for character inputs and process them, and read the ADC value when necessary (using ADCDataGet() and ADCIntStatus()) and display the appropriate results. For this the value displayed to the OLED had to be created and assigned to change with the clock cycles of the program, the interrupt status needs to be continually updated to not interfere with processes such as writing to the OLED (GRStringDraw, GrRectFill, etc.) and process the ADC with the potentiometer input.
6. A counter variable is included in this program which can be enabled and disabled by a user input. The timer frequency is also controlled using the TimerLoadSet() function with its third argument being SysCtlClockGet() for both timer 1 and 0. SysCtlClockGet() is in sysctl.c on line 2741 and it sets up the clock functionality and determines how many clock cycles the timer will wait before asserting an interrupt signal.
7. Configuring the display modes to the OLED was by far the most difficult part of this procedure. The display mode control is in the timer 0 ISR (displaying the requested number of operations per second and the serviced or actual number of operations per second) and uses UART to cycle through the display mode. It is necessary to not have any of the display mode controlling be done in the foreground so that it is not prone to starvation by high frequency interrupts.
8. Once it was ensured that the code all runs smoothly and as expected the sign offs were obtained and the lab was completed.

REPORT DISCUSSION

1. Functions and/or # defines used for the first time in this software project:
   a. #include “inc/hw_ints.h”
   b. #include “inc/hw_types.h”
   c. #include “driverlib/debug.h”
   d. #include “driverlib/interrupt.h”
   e. #include “driverlib/fpu.h”
   f. #include “driverlib/timer.h”
   g. #define floodPeriod
   h. #define parPeriod
   i. void initializations(void);
   j. void displayOLED(uint32_t a[3]);
   k. void menuSwitch(void);
2. Definitions:
   a. **Polled Input/Output**: A polled I/O is a loop iteration based I/O. While this method is safer it is costly in timing. Polling consistently checks the state of a device to see if it is ready to be utilized as seen in the code for blinky.
   b. **Interrupt Input/Output**: For an interrupt driven I/O the interrupt is only called when needed, saving time and memory. For interrupts the computer can spend time doing allocated tasks and only calling a specific procedure when necessary.
   c. **Vector Table**: A vector table regarding this laboratory exercise is the table of interrupts given in startup_ewarm.c. This table holds both the interrupt handlers and requests with an individual vector being the address of an interrupt handler for efficient calling and readability.
3. The Timer0 and Timer1 peripherals in this project use polled input/output and the UART, GPIO, and ADC peripherals use interrupt driven input/output.
4. The processor knows which function to execute as a service routine when an interrupt is pending because the function names are in the interrupt vector table where the processor goes to when an interrupt is pending. The only place, other than the function definition that the name of the function used as the service routines given in the source code of this software project is in starup_ewarm.c.
5. When the high-frequency timer (Timer 1) is running fast enough to starve the foreground process (in this case blinky) the 1 Hz timer (Timer 0) still executes because the ISR affects foreground tasks not the background timer interrupt tasks. Also, in this procedure Timer0 has a higher priority than Timer1 so Timer0 is executed before it is starved. If the timer roles were swapped, then the starvation would occur before the 1Hz timer and performance would be drastically affected. This is feasible under the conditions that ---.
6. Considering the frequencies 66MHz to 16MHz, the correlation between the recorded results and the operational frequency is that when the operation was CPU-bound the clock speed ran at a rate almost four times greater than the speed when the operation was IO-bound for both frequencies. This is because of the fact that the CPU-bound operation allows for the processing speed to directly increase with the sliding wiper of the potentiometer where as in an IO-bound operation the program must wait for the constant processing speed of, in this case, the OLED which is significantly lower.
7. In Figure 1.0 below, the function of each of the flag fields used as arguments to the ui32Config parameter of the SysCtlClockSet() is described.
   The parameters for the function using the frequency 16MHz (Line 106 in Appendix A):
   ```
   SysCtlClockSet(SYSCTL_SYSDIV_3 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN);
   ```
   The parameters for the function using the frequency 66MHz (Line 116 in Appendix A):
   ```
   SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
   ```



Note:

- **a.** Control provided by **RCC** register bit/field.
- **b.** Control provided by **RCC** register bit/field or **RCC2** register bit/field, if overridden with **RCC2** register bit USERCC2.
- **c.** Control provided by **RCC2** register bit/field.
- **d.** Also may be controlled by **DSLPCLKCFG** when in deep sleep mode.
- **e.** Control provided by **RCC** register SYSDIV field, **RCC2** register SYSDIV2 field if overridden with USERCC2 bit, or [SYSDIV2,SYSDIV2LSB] if both USERCC2 and DIV400 bits are set.
- **f.** Control provided by **UARTCC**, **SSICC**, and **ADCCC** register field.

<u>Figure 1.0 Main Clock Tree</u>

8. In this laboratory exercise I was surprised by the useful functionality of interrupts. I am used to using polled driven code and understanding the efficiency of interrupts is very insightful. One of the main setbacks was figuring out how to exactly integrate the timers into the code. The instructions for this laboratory seemed much lighter and more abstract than usual. This led to a huge delay in my program's completion. In the future I would most likely try to complete the program starting from the timers.c file instead of my previous laboratory C file to see if it would make the integration easier.

CONCLUSION

While this lab was significantly one of the toughest software programs in this course to integrate and run, the amount that was gained by the process was worth the effort. In the previous labs the takeaway was a skill that may not be used in everyday programming, ADC and UART, but for this laboratory exercise the knowledge and use of interrupts is a valuable skill to know for the future. Also, the first hand understanding of how simple changes to code can improve or degrade efficiency so drastically is a unique insight that is not able to be seen so materialistically in prior courses. In this lab I also drastically organized my code which made the procedure itself seem less confusing and the code itself more readable, which may sound obvious, but I was shocked at how putting some of the code into separate functions really changed the view of my software project. In conclusion, whether it was something miniscule or important, this exercise proved to be a very insightful project.

APPENDICES:

APPENDIX A: Lab Code

The code for this exercise is shown starting on the next page.

```c
//*****************************************************************************
//
// CS322.50 Labratory 4 Software File
// Developed by: Andrea Gray (c)
// Version: 1.5 19-FEB-2019
//
//*****************************************************************************

//*****************************************************************************
//
// Copyright (c) 2011-2017 Texas Instruments Incorporated.
//
// This is part of revision 2.1.4.178 of the DK-TM4C123G Firmware Package.
//
//*****************************************************************************
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/fpu.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/timer.h"
#include "grlib/grlib.h"
#include "drivers/cfal96x64x16.h"
#include "driverlib/uart.h"
#include "driverlib/adc.h"
#define blinkyOnPeriod 100000
#define blinkyOffPeriod 100000
#define scale 4095

//******************************************************************************
//
// Globals
//
//******************************************************************************
int jumpTick;
int color;
int whileLoop;
int clockTick;
int actualVal;
int RQTimerLoad;
int8_t shouldDisplayCounter;
int32_t blinkyHandler;
int32_t local_char;
uint32_t requestedVal[3];
char AV[50];
char CT[50];
tContext g_sContext;
tRectangle sRect;

//******************************************************************************
//
// Function Declarations
//
//******************************************************************************
void splash(void);
void blinky(void);
void clear(void);
void initializations(void);
void printMenu(void);
void putString(char *str);
void getADC(void);
void displayOLED(uint32_t a[3]);
void menuSwitch(void);

//*****************************************************************************
//
//! This example application demonstrates the use of the timers to generate
//! periodic interrupts. Each interrupt handler will toggle its own indicator
//! on the display.
//
//*****************************************************************************

int main(void) {

  //
  // Global Variable Intializations
  //
  shouldDisplayCounter = 0; // Maintains Timer1 OLED unless specified otherwise
  blinkyHandler = 1; // Maintains LED 'heartbeat' unless specified otherwise
  clockTick = 0; // Counts the number of cycles as a time keeper
  whileLoop = 1; // Maintains indefinite while loop unless program exits

  //
  // 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();
```

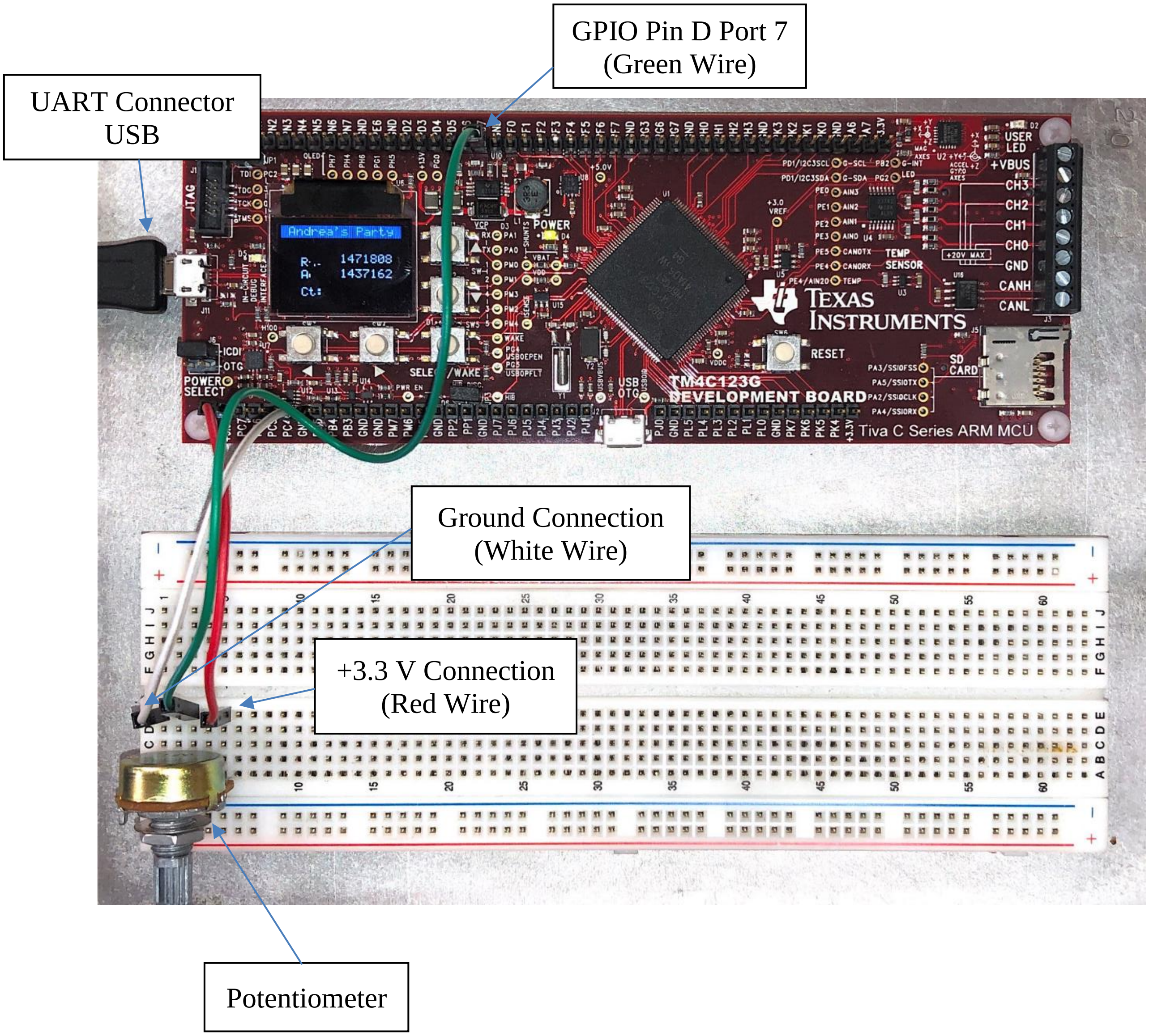
```c
  GrContextBackgroundSet(&g_sContext, ClrBlack);
  GrStringDrawCentered(&g_sContext, AV, -1, 68, 38, 1);
  clockTick = 0;
}

//*****************************************************************************
//
// The interrupt handler for the second timer interrupt.
//
//*****************************************************************************
void Timer1IntHandler(void) {
  TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT); // Clear the timer interrupt.
  clockTick++; // Increase clock count each second with the interrupt call.

  // Printing out the clock counter value if called for.
  if (shouldDisplayCounter == 1) {
    GrContextBackgroundSet(&g_sContext, ClrBlack);
    sprintf(CT,"    %d    ", clockTick);
    GrStringDrawCentered(&g_sContext, CT, -1, 68, 50, 1);
  }
}

//*****************************************************************************
//
// The UART interrupt handler.
//
//*****************************************************************************
void UARTIntHandler(void){
  uint32_t ui32Status;
  ui32Status = UARTIntStatus(UART0_BASE, true); // Get the interrupt status.
  UARTIntClear(UART0_BASE, ui32Status); // Clear the interrupt for UART

  //
  // Loop while there are characters in the receive FIFO.
  //
  while(UARTCharsAvail(UART0_BASE))
  {
    // Read the next character from the UART and write it back to the UART.
    UARTCharPut(UART0_BASE, UARTCharGet(UART0_BASE));
  }
}

//*****************************************************************************
//
// The function that is called when the program is first executed to set up
// the TM4C123G board, the peripherals, the timers, and the interrupts.
//
//*****************************************************************************
void initializations() {
  IntMasterDisable(); // Disable processor interrupts for configurations.
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG); // Enable GPIO G usage.

  //
  // Check if the LED peripheral access is enabled and wait if not.
  //
  while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOG)) {}
  GPIOPinTypeGPIOOutput(GPIO_PORTG_BASE, GPIO_PIN_2); // GPIO output is pin 2.
  CFAL96x64x16Init(); // Initialize the OLED display driver.

  // Initialize the OLED graphics context.
  GrContextInit(&g_sContext, &g_sCFAL96x64x16);
  GrContextFontSet(&g_sContext, g_psFontFixed6x8);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); // Enable UART 0 usage.
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); // Enable GPIO A usage.

  //  Set GPIO A0 and A1 as UART Pins.
  GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

  // configure uart for 115200 baud rate
  UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200, (UART_CONFIG_WLEN_8
                      | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
  clear(); // Clear any outputs or inputs from the PuTTY window.

  //*****************************************************************************
  //
  // 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.
  //
  //*****************************************************************************

  // Enable the peripherals for GPIOD and ADC.
  SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
  GPIOPinTypeADC(GPIO_PORTD_BASE, GPIO_PIN_7); // Set GPIO D4 as an ADC pin.
  ADCSequenceDisable(ADC0_BASE, 3); // Disable sample sequence 3.

  // Configure sample sequence 3: processor trigger, priority = 0.
  ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);

  //
  // Configure step 0 on sequence 3: channel 4. Configure the interrupt
  // flag to be set when the sample is done (ADC CTL IE). Signal last
```

```c
    // conversion on sequence 3 (ADC_CTL_END).
    //
    ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH4 | ADC_CTL_IE |
                             ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 3); // Enable sequnece 3.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0); // Enable usage of Timer 0.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1); // Enable usage of Timer 1.

    // Configure the two 32-bit periodic timers.
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
    TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet());
    TimerLoadSet(TIMER1_BASE, TIMER_A, SysCtlClockGet() / 10000);

    // Setup the interrupts for the timer timeouts.
    IntEnable(INT_TIMER0A);
    IntEnable(INT_TIMER1A);
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

    // Enable the timers.
    TimerEnable(TIMER0_BASE, TIMER_A);
    TimerEnable(TIMER1_BASE, TIMER_A);
}

//*****************************************************************************
//
// Gets input from ADC.
//
//*****************************************************************************
void getADC(){
    ADCProcessorTrigger(ADC0_BASE, 3); // Trigger the ADC conversion.
    while(! (ADCIntStatus(ADC0_BASE, 3, false))); //Wait for an ADC reading.
    ADCSequenceDataGet(ADC0_BASE, 3, requestedVal); // Put the reading into a var.

    // This adjusts the variable used for timer loading depending on if the
    // clock counter is enabled or not.
    if (shouldDisplayCounter == 1)
      RQTimerLoad = (1000*requestedVal[0])/4095;
    else
      RQTimerLoad = (2000000*requestedVal[0])/4095;

    // Printing the value being requested by the ADC peripheral.
    char RV[50];
    sprintf(RV,"    %d     ", RQTimerLoad);
    GrContextBackgroundSet(&g_sContext, ClrBlack);
    GrStringDrawCentered(&g_sContext, RV, -1, 68, 26, 1);
}

//*****************************************************************************
//
// 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 seperated only for report use -- multi-line strings are
    // not supported by C PL.
    char*menu = "\rMenu Selection: \n\rE - Erase Terminal Window\n\rL - Flash LED
         \n\rM - Print the Menu\n\rQ - Quit this program\n\rT - Enable Count\n\r";
    putString(menu);
}

//*************************************************************************
//
// If the input character is not invalid, begin the character matching
// statment below through the switch statment and act accordingly to
// the user input.
//
//*************************************************************************
void menuSwitch() {
    if (local_char != -1) {
      UARTCharPut(UART0_BASE, local_char); // Send a character to UART.
```

```
    // Begin character input matching to menu option.
    switch(local_char) {
    case 'E': // Clear PuTTY window
      clear();
      break;

    case 'L': //LED toggle
      if(blinkyHandler == 0)
        blinkyHandler = 1;
      else
        blinkyHandler = 0;
      GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_2, 0);
      break;

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

    case 81: // Quit program
      IntMasterDisable();
      putString("\n\rBYE!"); // Goodbye message to PuTTy
      // Re-draw OLED with goodbye statment in red font.
      sRect.i16XMin = 0;
      sRect.i16YMin = 0;
      sRect.i16XMax = GrContextDpyWidthGet(&g_sContext) - 1;
      sRect.i16YMax = GrContextDpyHeightGet(&g_sContext) - 1;
      GrContextForegroundSet(&g_sContext, ClrBlack);
      GrContextBackgroundSet(&g_sContext, ClrBlack);
      GrRectFill(&g_sContext, &sRect);
      GrContextForegroundSet(&g_sContext, ClrRed);
      GrContextFontSet(&g_sContext, g_psFontFixed6x8);
      GrStringDrawCentered(&g_sContext, "Goodbye", -1,
                           GrContextDpyWidthGet(&g_sContext) / 2, 30, false);
      whileLoop = 0;
      break;

    case 'T': // Display Counter
      if (shouldDisplayCounter == 0) {
        shouldDisplayCounter = 1;
      }
      else {
        sRect.i16XMin = 50;
        sRect.i16YMin = 45;
        sRect.i16XMax = 96;
        sRect.i16YMax = 64;
        GrContextForegroundSet(&g_sContext, ClrBlack);
        GrRectFill(&g_sContext, &sRect);
        shouldDisplayCounter = 0;
        GrContextForegroundSet(&g_sContext, ClrWhite);
      }
      break;

    }
  }
}

//*****************************************************************************
//
// Blinky LED "heartbeat" function.
//
//*****************************************************************************
void blinky() {
  if(blinkyHandler == blinkyOnPeriod) {
    GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_2, GPIO_PIN_2);
    blinkyHandler = -blinkyOffPeriod;
  }

  if(blinkyHandler == -1) {
    GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_2, 0);
    blinkyHandler = 1;
  }
}

//*****************************************************************************
//
// Splash Screen
//
//*****************************************************************************
void splash() {
  tRectangle Rect;
  int16_t xValLast = 0;
  int16_t yValLast = 38;
  int16_t xVal = 0;
  int32_t yVal = 38;
  while(1) {
    xVal+=2;
    if(jumpTick == -1) {
      jumpTick = 0;
    }

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

```
            if(jumpTick > 10) {
              jumpTick = -1;
            }
          }

          GrContextInit(&g_sContext, &g_sCFAL96x64x16);   // Sets OLED context.
          switch(color) {
          case 0:
            GrContextForegroundSet(&g_sContext, ClrBlue);
            break;
          case 1:
            GrContextForegroundSet(&g_sContext, ClrRed);
            break;
          case 2:
            GrContextForegroundSet(&g_sContext, ClrGreen);
            break;
          case 3:
            GrContextForegroundSet(&g_sContext, ClrBlack);
            sRect.i16XMin = 0;
            sRect.i16YMin = 0;
            sRect.i16XMax = xVal;
            sRect.i16YMax = GrContextDpyHeightGet(&g_sContext);
            GrRectFill(&g_sContext, &sRect);
            break;
          }

          GrCircleFill(&g_sContext, xValLast,yValLast, 5);
          GrContextForegroundSet(&g_sContext, ClrWhite);
          GrCircleFill(&g_sContext, xVal,yVal, 5);
          GrContextFontSet(&g_sContext, g_psFontCm12/*g_psFontFixed6x8*/);
          GrFlush(&g_sContext);
          xValLast = xVal;
          yValLast = yVal;

          if(xVal >= 106) {
            xVal = -13;
            color++;
          }

          if(color == 4) {
            break;
          }

          SysCtlDelay(refreshRate);
        }

        GrContextForegroundSet(&g_sContext, ClrBlack);
        Rect.i16XMin = 0;
        Rect.i16YMin = 0;
        Rect.i16XMax = 96;
        Rect.i16YMax = 64;
        GrContextForegroundSet(&g_sContext, ClrBlack);
        GrRectFill(&g_sContext, &Rect);
        GrContextForegroundSet(&g_sContext, ClrWhite);
        GrContextFontSet(&g_sContext, g_psFontFixed6x8);
        char clockRate[8];
        sprintf(clockRate, "%d Hz", SysCtlClockGet());
        GrStringDrawCentered(&g_sContext, clockRate, -1, 96 / 2, 40, 0);
        SysCtlDelay(SysCtlClockGet()/3);
        Rect.i16XMin = 0;
        Rect.i16YMin = 0;
        Rect.i16XMax = 96;
        Rect.i16YMax = 64;
        GrContextForegroundSet(&g_sContext, ClrBlack);
        GrRectFill(&g_sContext, &Rect);
        sRect.i16XMin = 0;
        sRect.i16YMin = 0;
        sRect.i16XMax = 96;
        sRect.i16YMax = 64;
        GrContextBackgroundSet(&g_sContext, ClrBlack);
        GrRectFill(&g_sContext, &sRect);
        GrContextForegroundSet(&g_sContext, ClrWhite);

        // Initialize timer status display.
        GrContextFontSet(&g_sContext, g_psFontFixed6x8);
        GrStringDraw(&g_sContext, "Rq:", -1, 16, 26, 0);
        GrStringDraw(&g_sContext, "Ac:", -1, 16, 36, 0);
        GrStringDraw(&g_sContext, "Ct:", -1, 16, 50, 0);

      }
```

APPENDIX B: TivaWare© Board Connections



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 ______________________

DK-TM4C123G Board Used 20

Lab Station Used 04

1. Demonstrate operation of your a project on the DK-TM4C123G board which is performing a variable number of interrupt service routine executions per second.

   Instructors's Initials DF Date/Time 2/27/19 11:20Am

2. Demonstrate a program which can display the number of times each second that the variable rate service routine is able to be performed together with switching between each of the following permutations.

	Number of ISR executions per second	
Operating Frequency	With an OLED display each variable frequency service call	Without an OLED display in variable frequency service call
16 MHz	216	415310
66 MHz	365	1473393

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 DF Date/Time 2/27/19 11:22Am

5 /5

APPENDIX D: Definitions

- **ADC—**Analog to Digital Converter
- **OLED**—Organic Light Emitting Diode on the TivaWare® TM4C123G Board
- **PDL**—Peripheral Driver Library
- **ISR**—Interrupt Service Routine

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: Feb 10, 2019].