	EMBRY-RIDDLE AERONAUTICAL UNIVERSITY **Department of Electrical, Computer, and Software Engineering** **CEC322**: Microprocessor Laboratory (Spring 2019), **SECTION** PC51 Due Date: 23-APR-2019		
	PREPARED BY:	**EQUIPMENT S/N:**	**PERFORMED:**
	Gray, Andrea Pietz, Daniel	DK-TM4C123G	16-APR-2019

LABORATORY #10

Playing a Musical Tune using the PWM Peripheral

PREFIX: DEFINITIONS

All new technical terms and acronyms are defined in this prefix, below.
PWM: Pulse-Width-Modulator

OBJECTIVES

- Increase knowledge and experience with the DK-TM4C123G Development Kit
- Continued use of the 'C' programming language and Timer peripherals
- Increases understanding of the engineering design process
- Introduces the use of the PWM peripheral on the TM4C123G
- Utilization of functions and constants in the TivaWare® Peripheral Driver Library

PURPOSE

The purpose of this laboratory exercise is to utilize the PWM peripheral of the TM4C microcontroller to play a musical tune accurately and efficiently.

SPECIFICATIONS

For this laboratory exercise, it is required to demonstrate to the appropriate faculty that the 'C' software project created in conjunction with the development board which is able to run under debug mode and completes the assignment requirements dictated below.
This program must use the PWM peripheral to generate a sequence of musical notes. The program must also be able to display a credits (splash) screen, play the musical tune upon start-up, and then terminate immediately following. It is also required to have a mechanism to restart the tune without using the reset button on the board or exiting the running code in debug mode.

PROCEDURE

The code is completed in the 'C' programming language with all sources listed in the process as they are used. The TivaWare® TM4C123G Development Kit is the only hardware utilized for this procedure.

Process:

1. Code Retrieval
   1.1. This program was built from the La9.c code © Andrea Gray 2019 for reliability and compiling reassurance purposes.
2. Overview
   2.1. The objective of this software is to perform the following tasks successfully in 'C'
      2.1.1. Splash screen as done in previous exercises
      2.1.2. Execution of a musical tune
         2.1.2.1. Played at start-up

2.1.2.2. Played when requested thereafter
2.1.2.3. No sound emitting between requests
2.1.3. I/O through UART to establish communication with the user and the program
2.1.3.1. Quit option
2.1.3.2. Blinky “heartbeat” toggle
2.1.3.3. Option to play the tune again, from the beginning

3. Gathering Information
   3.1. Information was gathered from the PWM example software project
      3.1.1. Configuring two (2) PWM signals
         3.1.1.1. Only one signal is configured and used in this project though
      3.1.2. Gathered from the Peripheral Driver Library
4. Timer Utilization
   4.1. The Timer in this project is used to call an ISR for each note of the musical tune
   4.2. Each time the interrupt is serviced, a different note will be played by the PWM peripheral
   4.3. ISR frequency is the notes-per-second of the music
5. Period and Width
   5.1. User configurable aspects of the PWM
      5.1.1. As discussed in the questions following, the period and width are both set up by the procedure each time that the function for playing a specific note is called
      5.1.2. The period and width are unique to the note that is being played
   5.2. The duty-cycle of the signal is a constant 50% in correspondence with procedural point five in [1]
      5.2.1. This means that the width is one-half, 50%, of the period as duty cycle is defined as width divided by the period
6. Frequencies
   6.1. The frequencies corresponding to each note in the tune are put into an array
   6.2. The frequencies were gathered from Figure 1 below.

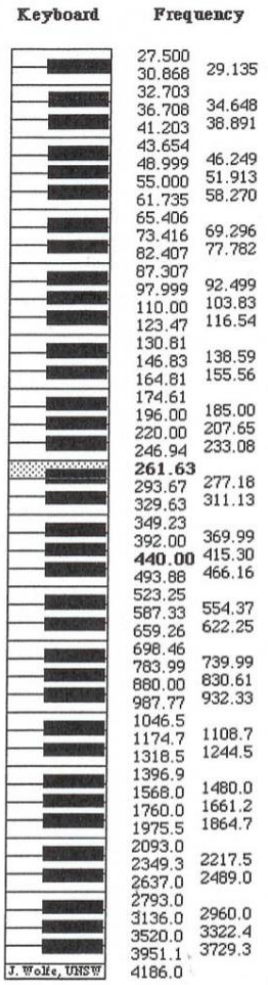

Figure 1: Musical Note Frequency Diagram

7. Playback
   7.1. The tune note number requirement was 32, and 133 notes were implemented
   7.2. Fur Elise was the song chosen for this exercise for its ease of recognition and simplistic melody
   7.3. The notes frequencies and period were put into separate arrays
      7.3.1. This was done rather than just a single array containing just the frequencies to allow for advantages on space and usage of more rhythmically complex pieces
      7.3.2. These arrays were gathered and created using a MATLAB script with Figure 1 that converts an MIDI sequence into a streamlined song composition array
   7.4. Playback of these notes are performed by a timer regulated ISR
8. Connections to Board
   8.1. Connections were made between the TM4C123G, PWM0 – PWM3 channel signals, and a headphone wire connecting channels PWM0 and PWM1.
      8.1.1. Headphone wire is connected to a speaker located at the work desk in the laboratory
         8.1.1.1. The speaker is ensured to be set to the lowest volume setting to begin with for obvious reasons

[Coming Soon to a Theater Near You]

Figure 2: Board Connections

9. OLED display
   9.1. The OLED display was maintained for

9.1.1. Splash Screen
9.1.2. Team name banner

9.2. The OLED display was altered to display the name of the song chosen, Fur Elise, upon song playback

REPORT DISCUSSION

1. Definitions:
   ⇒ **Pulse-Width Modulation**: A powerful technique for controlling analog circuits with a microprocessor's digital output [3]
   ⇒ **Frequency**: Occurrences divided by time. For example, in this exercise the frequency is defined as the number of times a waveform occurs over a one second time period
   ⇒ **Period**: The time of a single cycle in a continuous, indefinite event
   ⇒ **Duty Cycle**: The timing comparison ratio between the on period of a circuit and that same circuit's off period [4]
   ⇒ **Transfer Function**: A theoretical model of outputs for every input possible to a device [5]
2. Functions used for the first time in this laboratory exercise, including their name, included or linked files for the function, and a brief description, are described below.
   ⇒ GPIOPinTypePWM() in "driverlib/gpio.h" specifies that the GPIO pin being used is for Pulse-Width Modulation
   ⇒ PWMGenConfigure() in "driverlib/pwm.h" configures the mode of operation for a generator
   ⇒ PWMOutputState() in "driverlib.pwm.h" enables or disables the specified outputs
   ⇒ PWMGenDisable() in "driverlib/pwm.h" disables the timer for a generator block
   ⇒ PWMGenPeriodSet() in "driverlib/pwm.h" sets the period of the specified generator block
   ⇒ PWMPulseWidthSet() in "driverlib/pwm.h" sets the width of the pulses for a specific output
   ⇒ PWMGenEnable() in "driverlib/pwm.h" enables the timer for a generator block
3. It is necessary to calculate the period in CPU cycles for use in the TimerLoadSet() function call rather than SI units because the function is programming the processor to perform a certain task at a certain instant and the processor does not know what the human-time construct of seconds are. Rather, the processor does know the amount of cycles occurring over a given period and can count and determine those easily. These cycles can then be translated by the programmer to represent the desired SI unit.
4. The period specified in CPU as provided to a TimerLoadSet() function call considering an operating frequency of the microcontroller of 16 MHz and the desired frequency of a Timer ISR operation being at 20 kHz is shown below.
$$\frac{16\ MHz}{20\ kHz} = \frac{16{,}000{,}000}{20{,}000} = 800$$
5. The Timer peripheral cannot be used to generate exact and arbitrary frequencies. This is because, as seen in the following, the cycles are so large that miniscule changes and variations in numbers cannot be accurately represented by the operation. The calculation for the frequency representation corresponding to a period one CPU cycle shorter than 20 kHz and one CPU cycle longer than 20 kHz, as calculated in the question above, is shown below.
$$\frac{16{,}000{,}000}{19{,}999} = 800.0400\ldots$$
$$\frac{16{,}000{,}000}{20{,}001} = 799.9600\ldots$$
6. Consider: It is common for the designers of microcontrollers place multiple, even conflicting, functions onto a single pin.

⇒ This is done because doing such allows for different consumer groups to be targeted. This means that different users with different uses for the device can be satisfied by a single device and the company gets the most out of each pin.

⇒ The TivaWare® statement which tells the microcontroller how a certain pin is to be used by all of the functions on said pin is GPIOPinType???(). This configuration function can be used to configure multiple peripheral devices such as PWM, ADC, etc. Those acronyms, and many more found in [2] will be used to replace the ??? in the function for specific configurations.

7. Three other actuators that can be driven using a single pulse-width modulated signal, including whether they use the variable frequency, the variable pulse-width, or both to control the operation, are (1) RGB LED's which use both pulse-width and frequency modulation, (2) SERVO motors which again use both pulse-width and frequency modulation, and (3) Digital Communication which uses a fixed frequency with a pulse-width modulation. The transfer function for the actuator is shown below in Figure 3.

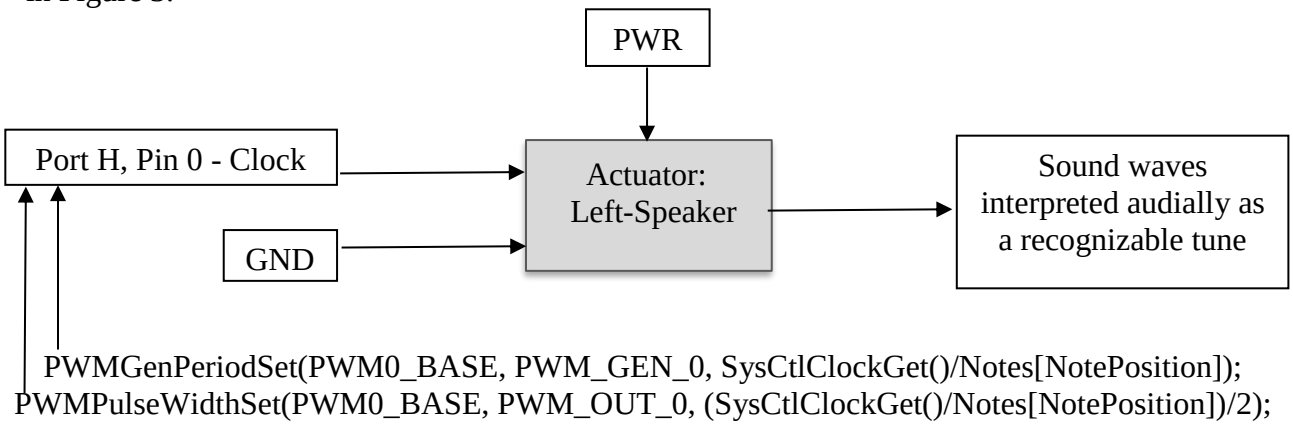

Figure 3: Actuator Transfer Function

8. Some surprises that were encountered in this lab were the lack of precision in the Timer peripheral frequencies and the difficulty associated with the PWM audial output. The lack of precision did create some less complex sounding tunes because the frequencies calculated were decimals (doubles) and the frequencies used by the functions were integers. There was no available solution to increase the precision in the song. The PWM was difficult to tweak in a way that lead to the sound emitting from the speaker to not only be accurate to the notes requested, but also to the timing including the silence required to follow immediately after the conclusion of the tune. For future projects and/or if this exercise were to be done again, it would be very interesting to see if multiple boards could be used and connected to create a more complex tune and to instantiate multiple arrays with an option to pick from different songs to play, much like an MP3-player.

CONCLUSION

In conclusion, this laboratory exercise was a bittersweet process. Although the completion of a course, especially a time-consuming course such as this, is always cause for celebration, there is still a small glimmer of sonder in the mix. The exercises encountered in this course were not only informative on practical applications and individual betterment of time-management, but they were also interesting and rewarding. This lab was successful and simplistic in its implementation and execution. The PWM is a new peripheral and the expected outcome was far from what has been done before. Nevertheless, this process was easy in that the code generation is routine at this stage, and the new functions were minimal and self-explanatory. Although the PWM is no Beethoven, and neither are we, we are proud to playback this tune of completion, achievement, and change; Fur Elise.

APPENDICES:

APPENDIX A: Lab Code

```
//*****************************************************************************
// File: Lab10.c
// Project: Lab10
// Author(s): Andrea Gray and Daniel Pietz
// Date Complete: April 23, 2019
//
// This application provides a connection from the TM4C board to the UART
// allowing the user to interact with the board via the keyboard. The
// application will also use an interrupt to send a values to the PWM to
// generate proper note frequencies in order to play a song.
//
//*****************************************************************************

#include <stdint.h> // Defines uint32_t, uint8_t
#include <stdbool.h> // Defines boolean values
#include <string.h> // Used for strlen()
#include <stdio.h> // Used for sprintf()
#include <ctype.h> // Used for toupper()


#include "inc/hw_ints.h" // Used to define Hardware Interrupts
#include "inc/hw_memmap.h" // Used to include constants involved with memory locations
#include "inc/hw_types.h" // Used in the original timers.c example



#include "driverlib/debug.h" // Debugging purposes
#include "driverlib/fpu.h" // Lazy stacking
#include "driverlib/interrupt.h" // Defines the interrupt Enable/Disable funtions
#include "driverlib/sysctl.h" // Defines anything prefixed with SysCtl ot SYSCTL
#include "driverlib/timer.h" // Defines Timer usage
#include "driverlib/pin_map.h" // Defines GPIO_PA0_U0RX and TX for the UART
#include "driverlib/gpio.h"
#include "driverlib/uart.h"
#include "driverlib/pwm.h" // Pulse-Width Modulator

#include "grlib/grlib.h" // Display
#include "drivers/cfal96x64x16.h" // OLED


#define LEDOn 100000 // defines how long the LED will stay lit
#define LEDOff 100000 // defines how long the LED will remain off

//*****************************************************************************
//
// Globals
//
//*****************************************************************************
int whileLoop = 1;
int32_t blinkyHandler = 1;
tContext Context;
tRectangle Rect;
uint32_t NotePosition = 0; // location in array
double Beats[] = {0,0.41129,0.79235,1.1411,1.4595,1.7506,2.0182,2.2654,2.495,2.9236,
                  3.1373,3.3509,3.5643,3.7776,4.2037,4.4164,4.6291,4.8415,5.0539,
                  5.4779,5.6896,5.9012,6.1124,6.3236,6.5343,6.745,6.9549,7.1648,
                  7.3732,7.5816,7.9926,8.1982,8.4039,8.6099,8.8164,9.2322,9.4428,
                  9.6567,9.8754,10.1017,11.1694,11.5091,11.8976,12.3522,12.7484,
                  13.1195,13.466,13.785,14.0627,14.4852,14.6965,14.9078,15.119,
                  15.3303,15.7528,15.9641,16.1754,16.3866,16.5979,17.0204,17.2317,
                  17.443,17.6543,17.8655,18.0768,18.2881,18.4993,18.7106,18.9219,
                  19.1331,19.5557,19.7669,19.9782,20.1895,20.4007,20.8233,21.0345,
                  21.2458,21.4571,21.6683,22.0909,22.3021,22.5134,22.7247,22.9359,
                  23.3636,23.5826,23.8044,24.0291,24.2567,24.7215,24.959,25.2005,
                  25.4464,25.6973,26.2196,26.4959,26.7885,27.1051,27.4553,28.2007,
                  28.8478,29.1477,29.4336,29.7064,29.967,30.2157,30.4523,30.6761,
                  30.8863,31.0837,31.281,31.4784,31.6758,31.8731,31.8731,32.0705,
                  32.4652,32.6626,32.86,33.0574,33.2547,33.6495,33.8468,34.0442,
                  34.2416,34.4389,34.8337,35.031,35.2284,35.4284,35.6285,35.8288,
                  36.0296,36.2313,36.4347,36.6406,36.8503,37.2857,37.5142,37.7516,
                  37.9993,38.2587,38.8196,39.1249,39.4499,39.7975,40.1711,41};
double Notes[] = {659.2551,622.254,659.2551,622.254,659.2551,493.8833,587.3295,
                  523.2511,440,220,261.6256,329.6276,440,493.8833,207.6523,329.6276,
                  415.3047,493.8833,523.2511,220,329.6276,659.2551,622.254,659.2551,
                  622.254,659.2551,493.8833,587.3295,523.2511,440,220,261.6256,
                  329.6276,440,493.8833,207.6523,329.6276,523.2511,493.8833,440,
                  659.2551,622.254,659.2551,622.254,659.2551,493.8833,587.3295,
                  523.2511,440,220,261.6256,329.6276,440,493.8833,207.6523,329.6276,
                  415.3047,493.8833,523.2511,220,329.6276,659.2551,622.254,659.2551,
                  622.254,659.2551,493.8833,587.3295,523.2511,440,220,261.6256,
                  329.6276,440,493.8833,207.6523,329.6276,523.2511,493.8833,440,
                  220,493.8833,523.2511,587.3295,659.2551,261.6256,391.9954,698.4565,
                  659.2551,587.3295,246.9417,349.2282,659.2551,587.3295,523.2511,220,
                  329.6276,587.3295,523.2511,493.8833,329.6276,659.2551,329.6276,
                  659.2551,659.2551,1318.5102,622.254,659.2551,622.254,659.2551,
                  622.254,659.2551,493.8833,587.3295,523.2511,523.2511,440,220,
                  261.6256,329.6276,440,493.8833,207.6523,329.6276,415.3047,493.8833,
                  523.2511,220,329.6276,659.2551,622.254,659.2551,622.254,659.2551,
                  493.8833,587.3295,523.2511,440,220,261.6256,329.6276,440,493.8833,
                  207.6523,329.6276,523.2511,493.8833,440,0}; // holds the notes
int32_t local_char;

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

```
94  //
95  // Function Declarations
96  //
97  //*****************************************************************************
98  void splash(void); // splash screen display function
99  void blinky(void); // "Heartbeat" function
100 void clear(void); // clear the PuTTy window
101 void initializations(void); // Sets-up the software and hardware for usage
102 void printMenu(void); // re-prints the menu options to PuTTy
103 void putString(char *str); // prints a string to the OLED
104 void menuSwitch(void); // Switches between menu options depending on the input
105 void PlayNote(void);
106
107 //*****************************************************************************
108 //
109 // The interrupt handler for the first timer interrupt.
110 //
111 //*****************************************************************************
112 void Timer0IntHandler(void) {
113   TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
114   PlayNote();
115
116   while(UARTCharsAvail(UART0_BASE)) {
117     local_char = UARTCharGetNonBlocking(UART0_BASE);
118     if (local_char != -1)
119       menuSwitch();
120   }
121 }
122
123 //*****************************************************************************
124 //
125 // This example application demonstrates the use of the timers to generate
126 // periodic interrupts.
127 //
128 //*****************************************************************************
129 int main(void) {
130   // Enable lazy stacking for interrupt handlers.  This allows floating-point
131   // instructions to be used within interrupt handlers, but at the expense of
132   // extra stack usage.
133   FPULazyStackingEnable();
134
135   // Set the clocking to run directly from the crystal
136   SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
137
138   initializations();
139   splash();
140   clear();
141   printMenu();
142   // Fill the part of the screen defined below to create a banner.
143   Rect.i16XMin = 0;
144   Rect.i16YMin = 0;
145   Rect.i16XMax = GrContextDpyWidthGet(&Context) - 1;
146   Rect.i16YMax = 9;
147   GrContextForegroundSet(&Context, ClrSlateGray);
148   GrRectFill(&Context, &Rect);
149   GrContextForegroundSet(&Context, ClrWhite);
150   GrStringDrawCentered(&Context, "Gray | Pietz", -1,
151                        GrContextDpyWidthGet(&Context) / 2, 4, 0);
152
153   IntMasterEnable();
154
155   // The main while loop the function will stay in unless acted upon by the
156   // user through UART or an interrupt timer is being serviced.
157   //
158   while(whileLoop != 0) {
159     // Calling the 'heartbeat' function if specified to do so.
160     if(blinkyHandler != 0) {
161       blinky();
162       blinkyHandler++;
163     }
164     whileLoop++;
165   }
166 }
167
168 //*****************************************************************************
169 //
170 // The error routine that is called if the driver library encounters an error.
171 //
172 //*****************************************************************************
173 #ifdef DEBUG
174 void __error__(char *pcFilename, uint32_t ui32Line) {}
175 #endif
176
177 //*****************************************************************************
178 //
179 // Using the character output function as a base for a parent function
180 // used to output an entire string to the OLED one character at a time.
181 //
182 //*****************************************************************************
183 void putString(char *str) {
184   for(int i = 0; i < strlen(str); i++) {
185     UARTCharPut(UART0_BASE, str[i]);
186   }
```

```
}

//*****************************************************************************
//
// Splash Screen
//
//*****************************************************************************
void splash() {
  // splash screen display parameters
  tRectangle screen;
  screen.i16XMin = 0;
  screen.i16XMax = 96; // Maximum width of the OLED
  screen.i16YMin = 0;
  screen.i16YMax = 64; // Maximum height of the OLED

  // clear the screen from any reminaing displays
  GrContextForegroundSet(&Context, ClrBlack);
  GrRectFill(&Context, &screen);

  // loading block parameters
  for(int length = 10; length <= 86; length++) {
    tRectangle loading;
    loading.i16XMin = 9;
    loading.i16XMax = length;
    loading.i16YMin = 26;
    loading.i16YMax = 39;
    GrContextForegroundSet(&Context, ClrSalmon);
    if (length%10 == 0) {
      GrStringDrawCentered(&Context, "Loading.  ", 11, 48, 20, true);
    }
    else if (length%10 == 1) {
      GrStringDrawCentered(&Context, "Loading.. ", 11, 48, 20, true);
    }
    else if (length%10 == 2) {
      GrStringDrawCentered(&Context, "Loading...", 11, 48, 20, true);
    }
    else {
      GrStringDrawCentered(&Context, "Loading   ", 11, 48, 20, true);
    }
    SysCtlDelay(150000);
    GrContextForegroundSet(&Context, ClrDeepSkyBlue);
    GrRectFill(&Context, &loading);
  }

  // clear the screen so that it is set for the main screen
  GrContextForegroundSet(&Context, ClrBlack);
  GrRectFill(&Context, &screen);
}



//*****************************************************************************
//
// 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.
    local_char = UARTCharGetNonBlocking(UART0_BASE);
    menuSwitch();
  }
}

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

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


//*****************************************************************************
//
// 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) { // Run only if the character in PuTTy is valid
    UARTCharPut(UART0_BASE, local_char); // Send a character to UART.

    // Begin character input matching to menu option.
    switch(local_char) {
    case 'C': // 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 'M': // Re-print menu
      UARTCharPut(UART0_BASE, 5);
      printMenu();
      break;

    case 'P':
      NotePosition = 0;
      break;

    case 'Q': // Quit program
      IntMasterDisable();
      putString("\n\rBYE!"); // Goodbye message to PuTTy
      Rect.i16XMin = 0;
      Rect.i16YMin = 0;
      Rect.i16XMax = GrContextDpyWidthGet(&Context) - 1;
      Rect.i16YMax = GrContextDpyHeightGet(&Context) - 1;
      GrContextForegroundSet(&Context, ClrBlack);
      GrContextBackgroundSet(&Context, ClrBlack);
      GrRectFill(&Context, &Rect);
      GrContextForegroundSet(&Context, ClrRed);
      GrContextFontSet(&Context, g_psFontFixed6x8);
      GrStringDrawCentered(&Context, "Goodbye", -1, GrContextDpyWidthGet(&Context)
                           / 2, 30, false); // Goodbye message to OLED in red.
      // If the user said to quit the whileLoop will NO LONGER be able to be ran
      whileLoop = 0;
      break;

    default:
      char invalid[25] = "\n\rInvalid. Try Again: ";
      char*ptr = invalid;
      putString(ptr);
      break;
    }
  }
}

//*****************************************************************************
//
// Sends a specific note to the PWM with a unique period and width
//
//*****************************************************************************
void PlayNote(void) {
  if (NotePosition < 134) { // if valid
    tRectangle loading;
    loading.i16XMin = 9;
    loading.i16XMax = 40;
    loading.i16YMin = 26;
    loading.i16YMax = 39;
    GrContextForegroundSet(&Context, ClrRed);
    GrStringDrawCentered(&Context, "Fur Elise", 11, 48, 20, true);
    IntMasterDisable();
    TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet()*(Beats[NotePosition+1] -
                 Beats[NotePosition])); // 120 BPM
    IntMasterEnable();
    PWMGenDisable(PWM0_BASE, PWM_GEN_0); // Disable The PWM generator to change values
    //setting period
    PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, SysCtlClockGet()/Notes[NotePosition]);
    //Set width for 50% duty cycle
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, (SysCtlClockGet()/Notes[NotePosition])/2);
    PWMGenEnable(PWM0_BASE, PWM_GEN_0); // start timers in gen 0
    NotePosition++; // incrementation to play the next note
  }
  else {
    tRectangle loading;
    loading.i16XMin = 9;
    loading.i16XMax = 40;
    loading.i16YMin = 26;
    loading.i16YMax = 39;
    GrContextForegroundSet(&Context, ClrRed);
    GrStringDrawCentered(&Context, "             ", 11, 48, 20, true);
  }
}

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

```
// 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(void) {
  IntMasterDisable(); // Disable processor interrupts for configurations.

  //*****************************************************************************
  //                                 LED
  //*****************************************************************************
  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.

  //*****************************************************************************
  //                                 OLED
  //*****************************************************************************
  CFAL96x64x16Init(); // Initialize the OLED display driver.
  GrContextInit(&Context, &g_sCFAL96x64x16); // Initialize OLED graphics
  GrContextFontSet(&Context, g_psFontFixed6x8); // Fix the font type

  //*****************************************************************************
  //                                 UART
  //*****************************************************************************
  SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); // Enable UART 0 usage.
  while(!SysCtlPeripheralReady(SYSCTL_PERIPH_UART0)) {}; // wait until UART 0 is ready
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); // Enable GPIO A usage.
  while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOA)) {}; // Wait until GPIO A is ready
  GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1); // Set Pins A0 and A1 for UART

  // Configure UART for 115200 baud rate, 8 in 1 operation
  UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200, (UART_CONFIG_WLEN_8 |
                      UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

  IntEnable(INT_UART0); // Enable the interrupt for UART 0
  UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);  // Enable specific UART interrupt

  //*****************************************************************************
  //                                 PWM
  //*****************************************************************************
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH); // enable GPIO port H
  SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0); // enable PWM 0
  GPIOPinTypePWM(GPIO_PORTH_BASE, GPIO_PIN_0); // assign port D Pin 5 to the ADC
  GPIOPinConfigure(GPIO_PH0_M0PWM0); // configure pin as PWm0
  PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
  PWMOutputState(PWM0_BASE, (PWM_OUT_0_BIT | PWM_OUT_1_BIT), true);

  //*****************************************************************************
  //                                 TIMERS
  //*****************************************************************************
  SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0); // Enabling Timer 0 for use

  // Configuring the 32-bit periodic timer 0
  TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
  TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet());

  IntEnable(INT_TIMER0A); // Enabling timer 0 for interrupt usage
  TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT); // Enabling timer 0 timeout
  TimerEnable(TIMER0_BASE, TIMER_A); // Enabling timer 0

  //*****************************************************************************
  //                             GPIO OUT SIGNAL
  //*****************************************************************************
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOL); // Enable GPIO Port L for usage
  while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOL)) {} // Wait for port to be ready
  GPIOPinTypeGPIOOutput(GPIO_PORTL_BASE, GPIO_PIN_0); // Enable GPIO PL0 for use
  GPIOPinTypeGPIOOutput(GPIO_PORTL_BASE, GPIO_PIN_1); // Enable GPIO PL1 for use
  GPIOPinTypeGPIOOutput(GPIO_PORTL_BASE, GPIO_PIN_2); // Enable GPIO PL2 for use
  GPIOPinTypeGPIOOutput(GPIO_PORTL_BASE, GPIO_PIN_3); // Enable GPIO PL3 for use

  //*****************************************************************************
  //                              MISCELLANEOUS
  //*****************************************************************************
  clear(); // Clear any outputs or inputs from the PuTTY window.
  GrFlush(&Context); // Flush any cached operations
}

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


//*****************************************************************************
//
// Print menu function that takes the complete menu as a string and
// utilizes the print string function created above to output the entire menu
```

```
465   // in one transmission block to the PuTTY window.
466   //
467   //*****************************************************************************
468   void printMenu() {
469     char*menu = "\rMenu Selection: \n\rC - Erase Terminal Window\n\rL - Flash LED\n\rM -
470                  Print the Menu\n\rP - Play Music\n\rQ - Quit this program\n\r";
471     putString(menu);
472   }
```

APPENDIX B: Instructor Sign Off

[Coming Soon to a Theater Near You]

APPENDIX C: Citation

[1] Dr. Brian Davis, *Laboratory Exercise #10*, Embry-Riddle Aeronautical University Prescott Department of Computer, Mechanical, and Software Engineering, 2019.

[2] *TivaWare™ Peripheral Driver Library: User's Guide.* Rev. 2016. [eBook]. Austin: Texas Instruments, 2013, p.354-355. Available: http://www.ti.com/lit/ug/spmu298d/spmu298d.pdf. [Accessed: 3-APR-2019].

[3] Barr, Michael, "Introduction to Pulse Width Modulation", *Embedded*, August 31, 2001. [Online]. Available: https://www.embedded.com/electronics-blogs/beginner-s-corner/4023833/Introduction-to-Pulse-Width-Modulation. [Accessed: 19-APR-2019].

[4] FLUKE, "What is duty cycle?", *fluke.com.* [Online]. Available: https://www.fluke.com/en-us/learn/best-practices/measurement-basics/electricity/what-is-duty-cycle. [Accessed: 20-APR-2019].

[5] Wikipedia, "Transfer Function", *wikiedia.com*. [Online], Available: https://en.wikipedia.org/wiki/Transfer_function. [Accessed: 20-APR-2019].