<table>
<tr><td rowspan="3"></td><td colspan="3">EMBRY-RIDDLE AERONAUTICAL UNIVERSITY<br>Department of Electrical, Computer, and Software Engineering<br>CEC322: Microprocessor Laboratory<br>(Spring 2019), SECTION PC51<br>Due Date: 16-APR-2019<br>Turned in: 16-APR-2019</td></tr>
<tr><td>PREPARED BY:</td><td>EQUIPMENT S/N:</td><td>PERFORMED:</td></tr>
<tr><td>Gray, Andrea<br>Grobmeier, Drew</td><td>DK-TM4C123G<br>ARM 7.70.1</td><td>09-APR-2019</td></tr>
</table>

# LABORATORY #9

Using Assembly Language to Accelerate the Operation of an ISR

## OBJECTIVES

- Increase knowledge and experience with the DK-TM4C123G Development Kit
- Continued use of the ‘C’ programming language
- Increases understanding of the engineering design process
- Write ARM assembly language routines
- Use assembly language to accelerate a critical portion of the software project
- Utilize interrupts to generate high frequency operation

## PURPOSE

The purpose of this laboratory exercise is to combine the structure and usage of both the Assembly and C languages in one program to accelerate the portion of a program with the most expensive usage of CPU cycles in terms of fractional consumption. It is also a goal of this procedure to correctly and accurately write, develop, and assembly software using UAL.

## 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 show the accurate implementation assembly language code written to replace the variable frequency interrupt service routine (ISR) previously developed in ‘C’ in the fourth laboratory exercise. The sped-up operation must be displayed to validate operation of the program.

## 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 and a potentiometer are the only hardware devices utilized for this procedure. All new technical terms and acronyms are defined in Appendix D at the end of this report.

### *Process:*

1. Code Retrieval
   1.1. This program was built from the Lab4.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. Display ADC data
      2.1.2. Switch between 16 and 80 MHz
      2.1.3. Splash screen as done in previous exercises
      2.1.4. I/O through UART to establish communication with the user and the program

2.1.4.1. Quit option

2.1.5. Blinky "heartbeat"

2.2. The following task must be performed and integrated successfully in Assembly Language Code

2.2.1. Incrementation of the serviced counter

3. Translation

3.1. Since the only part of the code that needed to be changed was the clock frequency and the ISR for Timer 1, the procedure for this exercise is concise

3.2. Clock frequency options were changed from 16 and 60 MHz to 16 and 80 MHz

3.3. `Timer1IntHandler()` was reduced in lines to allow assembly translation to be easier

3.3.1. Printing of data was allocated to `Timer0IntHandler()`

3.3.2. The final `Timer1IntHandler()` is shown below in Figure 3.3.2.1

```
void Timer1IntHandler(void) {
  TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT); // Clear the timer interrupt.
  ServicedCount++; // Increase serviced count each second with the interrupt call.

}
```

**Figure 3.3.2.1: Timer 1 Interrupt Handling Function – 'C'**

3.4. A breakpoint was inserted at the function above to use the disassembly feature of IAR to gather the assembly code of the ISR.

4. Vector Table Changes

4.1. The new interrupt service routine had to be implemented into the `startup-ewarm` ISR vector table to allow the function to be properly called

4.2. Declared both `Timer1IntHandler()` and `isr_asm_start.s` at the beginning of the `startup-ewarm`

4.2.1. Manually changed the value in the vector table to switch between the assembly and C version of the interrupt service routine for timer 1

## REPORT DISCUSSION

1. The fastest frequency of operation observed was 3,029,675 Hz. The clock period of this frequency was 25 CPU cycles, which was calculated by $CPU\ Cycles = \frac{SysCtlClockGet()}{ADC\ Value}$ and was represented as the period value in the code and on the sign-off in the appendices.
2. There are seven instructions, from start to finish, in the post-assembled version of the assembly language routine created for this project.
3. Using the frequency, period, and instruction count from the questions above, the number of cycles per instruction for the variable frequency ISR, when operating at maximum frequency, is calculated below.

$$\frac{Time}{Program} = \frac{Instructions}{Program} * \frac{Cycles}{Instruction} * \frac{Time}{Cycle}$$

$$\frac{1}{3029675} = \frac{12}{1} * \frac{Cycles}{Instruction} * \frac{1}{80000000}$$

$$\frac{Cycles}{Instruction} = \frac{80000000}{3029675} * \frac{1}{12}$$

$$\frac{Cycles}{Instruction} = 2.20$$

4. The Assembly ISR laboratory exercise, this exercise, is 1,154,086 executions per seconds faster than the Timer Lab results. This means that the implementation of assemble language into the same general program speeds up execution by 161.53 %.

5. The machine language equivalent of `pop {pc}` in assembly language, usually the last instruction in the language, is `0x21bc : 0xbd00 POP {pc}`.
6. In this exercise, in addition to the use of assembly language, the techniques used to accelerate the incrementation in this project are listed below
   ⇒ Optimization of Assembly Code
   ⇒ Optimization of C code
   ⇒ Wrote the `TimerIntClear()` function in Assembly instead of branching to it
7. A surprise in this exercise was that the period changed between 22 and 24 between operational frequency changes for the assembly language implementation. Dr. Davis was initially surprised to see this outcome as well and suggested that our optimizations may not be complete due to this reaction. It was challenging in this exercise to implement the assembly. Although not completely new to either of the authors, the correct implementation and translation of the language proved to be still quite a new topic for both Drew and Andrea. The bugs and errors that were encountered and overcome in the laboratory four exercise, which this procedure was based from, came back into light this time around and since there has been some time between exercises four and nine, they proved to be just as confusing now as they were then. In the future it would be suggested that the students look over the laboratory exercise four again to become reacquainted with its functions and procedures; just to make the implementation of the assembly code a little bit easier.

# CONCLUSION

In conclusion, this laboratory exercise, while still not mastered by either author, was quite easy in regards to their history with assembly language. The code took the `Timer1IntHandler()` from laboratory four exercise and rewrote it in assembly language, then implemented it back into the exercise. The function `isr_asm_start()` was used to replace the `Timer1IntHandler()` ISR call. For both cases of 16 MHz and 80 MHz the program proved to be greatly accelerated by the instantiation of the assembly language code.

# APPENDICES:

## *APPENDIX A: 'C' Lab Code*

```
//*****************************************************************************
// Authors: Andrea Gray and Drew Grobmeier
//
// CEC322 Lab #9: sdf
// Development reference: Lab4.c
//
//*****************************************************************************
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>

#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"

#include "grlib/grlib.h"

#include "drivers/cfal96x64x16.h"

#include "driverlib/uart.h"
#include "driverlib/adc.h"
```

```c
#include "driverlib/gpio.h"
#include "driverlib/fpu.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/timer.h"
#include "driverlib/debug.h"

#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;// Maintains indefinite while loop unless program exits
int actualVal;
int ADCLoadValue;
int ServicedCount = 0;

int32_t BlinkyToggle = 1;// Maintains LED 'heartbeat' unless specified otherwise
int32_t CharacterInput; // Keeps the character input into PuTTy

uint32_t ADCValue[3];

char ServicedValue[50];
char PeriodValue[50];

tContext Context; // OLED drawing contextual structuring
tRectangle sRect; // Rectangle parameters for banner structuring

//*****************************************************************************
//
// Function Declarations
//
//*****************************************************************************
void splash(void); // splash screen display function
void blinky(void); // "Heartbeat" function
void clear(void); // clear the PuTTy window
void initializations(void); // Sets-up the software and hardware for usage
void printMenu(void); // re-prints the menu options to PuTTy
void putString(char *str); // prints a string to the OLED
void menuSwitch(void); // Switches between menu options depending on the input
void getADC(void); // Reading the value from the ADC

//*****************************************************************************
//
// The error routine that is called if the driver library encounters an error.
//
//*****************************************************************************
#ifdef DEBUG
void __error__(char *pcFilename, uint32_t ui32Line) {}
#endif

//*****************************************************************************
//
// The interrupt handler for the first timer interrupt.
//
//*****************************************************************************
void Timer0IntHandler(void) {
  TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); // Clear the timer interrupt.
  getADC(); // Gather values obtained by the ADC

  // setting the load value
  TimerLoadSet(TIMER1_BASE, TIMER_A, (SysCtlClockGet()/ ADCLoadValue));

  // Printing out the values that have been serviced.
  sprintf(ServicedValue,"Srv: %d          ", ServicedCount);
  GrContextBackgroundSet(&Context, ClrBlack);
  GrStringDraw(&Context, ServicedValue, 20, 5, 38, 1);


  // Printing out the ADC requested value if called for.
  GrContextBackgroundSet(&Context, ClrBlack);
  sprintf(PeriodValue,"Per: %d          ", SysCtlClockGet()/ADCLoadValue);
  GrStringDraw(&Context, PeriodValue, 20, 5, 50, 1);

  ServicedCount = 0;
}

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

}

//*****************************************************************************
//
// ADC Input Gathering
//
//*****************************************************************************
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, ADCValue); // Put the reading into a var.
  ADCLoadValue = (ADCValue[0]* (SysCtlClockGet() / 80000) +1);

  // Printing the value being requested by the ADC peripheral.
  char RequestedString[50];
  sprintf(RequestedString,"Req: %d          ", ADCLoadValue);
  GrContextBackgroundSet(&Context, ClrBlack);
  GrStringDraw(&Context, RequestedString, 20, 5, 26, 1);
}
```

```c

//*****************************************************************************
//
// The UART interrupt handler.
//
//*****************************************************************************
void UARTIntHandler(void){
  uint32_t ui32Status; // Holds the interrupt status
  ui32Status = UARTIntStatus(UART0_BASE, true); // Get the interrupt status.
  UARTIntClear(UART0_BASE, ui32Status); // Clear the interrupt for UART
  while(UARTCharsAvail(UART0_BASE)) { // Loop while there are characters in the receive FIFO.
    CharacterInput = UARTCharGetNonBlocking(UART0_BASE); // Read the next char from UART and write it back
    menuSwitch(); // Act accordingly via user request
  }
}

//*****************************************************************************
//
// 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) {

  //*****************************************************************************
  //                                    LED
  //*****************************************************************************
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG); // Enable GPIO G usage.
  while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOG)); // Wait until LED is ready
  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));

  //*****************************************************************************
  //                                  ADC
  //
  // 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.
  //*****************************************************************************
  SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0); // Enable ADC0
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD); // Enable GPIO D
  GPIOPinTypeADC(GPIO_PORTD_BASE, GPIO_PIN_7); // Set GPIO D7 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
  // 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.

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

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

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

```
//
// 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++) { // Loop through the string
    UARTCharPut(UART0_BASE, str[i]); // print each individual character
  }
}

//*****************************************************************************
//
// 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() {
  char*menu = "\rMenu Selection: \n\rC - Erase Terminal Window\n\rL - Flash LED\n\rM - Print the Menu\n\rQ - Quit this program\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 (CharacterInput != -1) { // Run only if the character in PuTTy is valid
    UARTCharPut(UART0_BASE, CharacterInput); // Send a character to UART.

    switch(CharacterInput) { // Begin character input matching to menu option.

    case 'C': // Clear PuTTY window
      clear();
      break;

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

    case 'M': // Re-print menu
      UARTCharPut(UART0_BASE, 5);
      printMenu();
      break;

    case 'Q': // Quit program
      IntMasterDisable();
      putString("\n\rBYE!"); // Goodbye message to PuTTy
      sRect.i16XMin = 0;
      sRect.i16YMin = 0;
      sRect.i16XMax = GrContextDpyWidthGet(&Context) - 1;
      sRect.i16YMax = GrContextDpyHeightGet(&Context) - 1;
      GrContextForegroundSet(&Context, ClrBlack);
      GrContextBackgroundSet(&Context, ClrBlack);
      GrRectFill(&Context, &sRect);
      GrContextForegroundSet(&Context, ClrRed);
      GrContextFontSet(&Context, g_psFontFixed6x8);
      GrStringDrawCentered(&Context, "Goodbye", -1, GrContextDpyWidthGet(&Context) / 2, 30, false); // Goodbye message to OLED in red.

      whileLoop = 0; // If the user said to quit the whileLoop will NO LONGER be able to be ran
      break;

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

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

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

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



//*****************************************************************************
//
// Main function
//
//*****************************************************************************
int main(void) {
    // 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();

    //*************************************************************************
    //
    // Checking if #define is set for the use of 66MHz for the Clock frequency.
    //
    //*************************************************************************
    #define MHz80 // Uncommented for use of 66MHz, Commented otherwise.

    #ifdef MHz80
    // Setting the clock to run directly from the crystal.
    SysCtlClockSet(SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);


    #else // If not using 66MHz, then set to 16MHz
    // Setting the clock to run directly from the crystal.
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);

    #endif

    IntMasterDisable(); // Disable interrupts while set up is in progress

    // Calling the initial functions to set up TM4C123G, its peripherals,
    // OLED and clear PuTTy window for menu output.
    initializations();
    splash(); // Prints out the splash screen to OLED.
    clear(); // Clears the PuTTy window for neatness.
    printMenu(); // Prints the UART menu for user IO.

    // Fill the part of the screen defined below to create a banner.
    sRect.i16XMin = 0;
    sRect.i16YMin = 0;
    sRect.i16XMax = GrContextDpyWidthGet(&Context) - 1;
    sRect.i16YMax = 9;
    GrContextForegroundSet(&Context, ClrSlateGray);
    GrRectFill(&Context, &sRect);
    GrContextForegroundSet(&Context, ClrWhite);
    GrStringDrawCentered(&Context, "00010000 01000000", -1, GrContextDpyWidthGet(&Context) / 2, 4, 0);

    IntMasterEnable(); // Enables Interrupts

    //************************************************************************
    //
    // The main while loop the function will stay in unless acted upon by the
    // user through UART or an interrupt timer is being serviced.
    //
    //************************************************************************
    while(whileLoop != 0) {
      // Calling the 'heartbeat' function if specified to do so.
      if(BlinkyToggle != 0) {
        blinky();
        BlinkyToggle++;
      }

      // Checking to see if the user has input a character and acting accordingly.
      while(UARTCharsAvail(UART0_BASE)) {
        CharacterInput = UARTCharGetNonBlocking(UART0_BASE);
        menuSwitch();
      }
    }
}
```

## *APPENDIX C: Assembly Lab Code*

```
#include "inc/hw_memmap.h"
#include "inc/hw_timer.h"
  name isr_asm
  section .text:CODE
  extern ServicedCount
  public isr_asm_start
isr_asm_start:
  push{lr};

  MOV   R1, #1
  MOV32 R0, TIMER1_BASE
  STR   R1, [R0, #0x24]

  MOV32 R0, ServicedCount
  LDR   R1, [R0]
  ADD   R1, R1, #1
  STR   R1, [R0]

  pop {pc} ; return
  end
```

## 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 Drew Grobmeier

TM4C Board Used 28

Lab Station Used 09

1. Demonstrate operation of a software project which calls an assembly language written subroutine. It is anticipated this will be the execution of an assembly language interrupt service routine as described herein, but execution of any assembly language written routine is sufficient for this signoff.

   Instructors's Initials ______ Date/Time ______

2. Demonstrate operation of a software project which meets all of the requirements set for in the laboratory procedure. This is execution of a high speed interrupt service routine written in assembly language. At least (2) frequencies must be evaluated for each version ISR, note if the frequencies evaluated are different than those given below..

| | Number of ISR executions per second | | |
|---|---|---|---|
| Operating Frequency | ISR Written in 'C' | ISR Written in Assembly | |
| 16 MHz | 163562 | 199300 | 676E3 P=22 |
| 80MHz | 1875589 | 3029675 | P=25 |

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

Instructors's Initials ______ Date/Time 15-APR 4:49pm

All Reqmts Met

5 /5

## APPENDIX D: Definitions

**I2C**: Inter-Integrated Circuit
**UAL:** Unified Assembly Language

## APPENDIX D: Citation and Resources

[1] *Tiva™ TM4C123GH6PM Microcontroller: Data Sheet*. Rev. E. [eBook]. Austin: Texas Instruments, 2014, p.898. Available: http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf. [Accessed: 3-APR-2019].

[2] Dr. Brian Davis, *Laboratory Exercise #9*, Embry-Riddle Aeronautical University Prescott Department of Computer, Mechanical, and Software Engineering, 2019.

[3] *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].