EEL 4745C: Microprocessor Applications 2
Lab ①: Basic Interfacing, Linking, and Communication
Fall 2022

OBJECTIVES
In this lab, you will interface an LED driver with the TIVA launchpad by following its I2C communication protocol. You will write some driver modules as well as I2C functions to complete and demonstrate your full understanding. You will also learn to link assembly functions to your C code, particularly for checksum validation of integer data. Lastly, you will experiment with the basics of UART communication and configure console I/O for Tiva in CCS.

REQUIRED MATERIALS
Hardware
- Tiva TM4C123GH6PM Launchpad
- LP3943 LED array modules (that are already) integrated to your daughterboard
- Laptop workstation with CCS setup

Software
- TivaWare and Board Support Package (BSP)
- Skeleton code provided for Lab 1

Documentations
- TM4C123GH6PM manual and datasheet
- LP3943 manual and datasheet
- Lecture 2 contents
- Feel free to use books or browse the internet!

Wiki++
- https://en.wikipedia.org/wiki/Magic_square
- https://en.wikipedia.org/wiki/Fletcher's_checksum
- https://en.wikipedia.org/wiki/uart
- https://en.wikipedia.org/wiki/I%C2%B2C

Part A: Interfacing LED drivers, I2C communication

MOTIVATIONS
When working with embedded systems, you will need to interface with external devices. Communication with external devices can be parallel or serial in nature. Serial methods of communication are extremely common because they decrease the number of lines or traces that need to connect the master and slaves. During this semester, and for years to come, you will find I2C useful because a very large number of devices can be addressed using only two lines or traces. This exercise will introduce you to the registers used by the TM4C123 for I2C communications, and prepare you to interface with external devices.

PREPARATION
- If you haven't read the manual for the LP3943 and the I2C section of the TM4C123GH6PM manual, stop and do so now.
- See the **Lecture 2** materials thoroughly

LED DRIVER BASICS
We will be controlling 16 RGB LEDs, which means that we will be powering 48 individual LED channels. This is a major reason to use an LED driver, we don't lose 48 GPIO channels for LEDs. Instead, we will be using three LED drivers, one for the red, blue, and green channels of each LED.

The LED drivers we have chosen are not just a simple driver, they also are small microcontrollers that can generate a PWM signal on each of their 16 LED channels. This is very useful because we can use the duty cycle of the PWM signal to dim or brighten each channel. In effect this will allow us to create any visible color by combining different brightness values of the red, green, and blue channels of each LED. This is the same process used by your phone, tablet, and computer screens. You are not required to use the LEDs in PWM mode, but **+1 bonus point** if you do so!

ASSIGNMENT
You will implement some functions for the LED driver library (we provided) to establish I2C communication with the LP3943s on your board. Your code should have all detailed instructions on which functions to implement and how! Please understand the given library functions and skeleton provided to you. In particular, pay attention to these files/functions:
- main.c, I2CDriver.c, and I2CDriver.h
- Board Support Package > I2C.h and I2C.c
- And cross-reference the functions or variables called by called/used from these scripts

PRE-LAB QUESTIONS
- What is the maximum clock speed of the LP3943?
- In your own words, sketch and describe the I2C communication process for the LP3943

IN-LAB REQUIREMENTS
Complete the LED driver and I2C functions outlined in the given code and run your code on any (16-bit) integer and display the result in LEDs.

Part B: Linking assembly functions to drive the LEDs

MOTIVATIONS
Writing assembly code is needed when creating low level performance-critical code. In Lab 2, you will be writing the core sections of the G8RTOS operating systems. Some of these sections will need to be written in assembly. This lab will help you familiarize yourself with ARM **Thumb2 Assembly** programming; in particular, you will implement the useful concept of Fletcher's 16-bit checksum algorithm on a predefined 2D array, *ie*, a 3x3 or 4x4 magic square.

Moreover, *checksums* are widely used to check if a data transfer was valid. Learning checksum algorithms are extremely important in the world of digital data communication and embedded computing. This is even more important for space embedded computing because solar radiation is known to flip random bits during a data transfer process.

CHECKSUM BASICS
There are many types of algorithms for computing checksum: fast algorithms use bitwise operations (*eg*, parity, xor, additions); moderately fast algorithms (eg, Fletcher, Alder); and slow algorithms (eg, CRS - cyclic redundancy checks). In particular, Fletcher checksum works as follows:

- Define two temporary variables: *sum1* and *sum2* that are twice as large as the data type
- Accumulate *sum1* over the data set
- Each time *sum1* is updated, the new value of *sum1* is added onto *sum2*
- To assure no overflow, use a modulo 255 operation each time a sum is computed
- Return a concatenation of sum2 (MSBs) and sum1 (LSBs)

See the **Lecture 2** slides for more details!

ASSIGNMENT
You need to write an assembly function that implements the Fletcher-16 algorithm and a C main function that calls it and compares the output to a C implementation. The final output needs to be shown as a 16-bit number in the LEDs.

The main.c program should create a 4x4 or 3x3 magic square matrix of uint8_t data type. See the template code provided for your convenience; please follow the guidelines to mimic the C version and complete the corresponding assembly function. Be careful how you address the individual bytes in assembly (look at the instruction set carefully). To help you get started, the modulo 255 function is already implemented (callable C function) and the whole main.c pipeline is given to you as a skeleton code.

Once you get the LED output, you can check whether the comparison is correct or not. Both the C and assembly function should give the same output, which should match with your calculations!

PRE-LAB QUESTIONS

- Given an array of integers, can you compute the Fletcher-16 checksum simulation yourself?

IN-LAB REQUIREMENTS
Complete the assembly function implementation and demo Fletcher checksum for a magic matrix in the LEDs as the output; a different matrix will be provided in the lab.

Part C: Basic UART with LED driver and console I/O

MOTIVATIONS
UART communication was a major focus in your previous (uP) course, so we will not spend too much time on this. However, UART concepts are very important to learn and implement for numerous embedded application scenarios. In particular for you, this will help you quickly debug and display content of variables rather than tracking register values and so on. Besides, we will use UART to display sensory outputs in future lab assignments as well.

ASSIGNMENT
Complete the UART initialization function provided in the code template; then configure the console COM port correctly to establish communication. Use built-in library functions such as the UARTprintf for validation.

I2C vs UART BASICS
I2C (Inter-Integrated Circuit) is a synchronous communication protocol, used to transfer data using a *master-slave (bidirectional)* framework. It is a two-wire serial bus that uses serial clock (SCL) and serial data (SDA) wires to send and manage data bit by bit between devices connected to the bus.

Unlike I2C, UART (Universal Asynchronous Receiver/ Transmitter) is a physical circuit that incorporates two devices for serial asynchronous communication. It doesn't operate using a clock so it is necessary for the baud rates of each UART to be within 10% of each other to prevent data loss. Moreover, it requires *Tx-Rx (ie, transmission-receiver)* agreement between the two devices ahead-of-time.

PRE-LAB QUESTIONS

- What are the major differences and similarities between I2C & UART communication?
- In which cases you will use one over the other?

IN-LAB REQUIREMENTS
Demo the same Fletcher checksum for a magic matrix in the console!
