# EELE465 Lab 2

Ryan French
ryanfrench3@montana.edu

Montana State University — April 20, 2020

## Introduction

The MSP430FR2310 is a 16-bit RISC architecture microcontroller with 11 GPIO pins. This microcontroller is of the same family as the MSP430FR2355 Launchpad, which means that we can use the on-board programmer to flash firmware onto the chip. Setting up I2C communication between the two microcontrollers allows the MPS430FR2355 to be used as a master which can read input from a membrane keypad and the MSP430FR2310 to be used as a slave device that controls a system of 8 LEDs. Input from the keypad shall be used to select from four distinct LED patterns.

## Setup

Setting up I2C connections is simple, as one only needs to run two wires with 10k pull-up resistors, which are placed close to each of the microcontrollers. Because of the limited current output of the GPIO pins, a flip-flop must be used to control the output to the LED bar. One of the master GPIO pins was attached to the slave's reset pin, as there was a consistent hang-up while switching between modes (this is something that will be diagnosed and fixed in the future).
The master was connected to the membrane keypad through eight GPIO pins. The driven pins were connected to a quad or-gate, which was used simply to monitor an interrupt, which occurs when any of the buttons is pressed. See the Eagle schematic for the full circuit.

## Solution

### Master Routines

The master in the circuit was programmed under the Energia IDE (a branch of Arduino), so it was easy to implement I2C communication with the native Wire library. Reading the keypad with the master involved a simple algorithm, where the column pins were designated as outputs, and rows were designated as inputs. With the column pins high, the main loop scanned the interrupt pin until it went high. The entire algorithm is shown below:

**Algorithm:** Read Key Press

```
Clear Column Pins;
for column pins do
    Set Column Pin;
    for row pins do
        Read Pin;
        if read pin then
            add_button_press();
        end
    end
    Clear Column Pin;
end
Set Column Pins;
Send I2C data;
```

As shown, the algorithm is simple and very quick. Custom digital pin read/write functions were written because of the large overhead of the methods included with Energia.

### Slave Routines

Something that isn't immediately obvious for the MSP430FR2310 is the need to unlock LPM5, which is to say that all pins must be unlocked from their high-impedance mode. Energia doesn't yet support this microcontroller, so all developing was done in Code Composer Studio. Setting up I2C is made easier with a CCS library called driverlib, which is a high-level library for setting up different communication protocols. Using these methods, the SDA and SCL were set up as I2C and an interrupt was attached.

The main loop is simple, it just constantly scans for a global flag which identifies that I2C data is available to use. A switch-case statement handles this data, selecting the LED pattern based on the ASCII data received. The patterns are generated using a custom function that takes a byte and uses it as a bitmask. Each bit is cycled through and the corresponding LED pin is written/cleared. The writing process is done through the flip-flop, so all 8 of the LED pins were set and the F-F clock was toggled to output from the flip-flop. The slave's code is detailed in flowchart 1.

## Comments

This lab went pretty smoothly, but the only issue was setting up I2C protocol on the MSP430FR2310. The datasheets weren't entire clear on how to accomplish this, and the example code that I could find didn't work. After messing around with interrupts and register offsets, I finally was able to get everything to work.

# Appendix

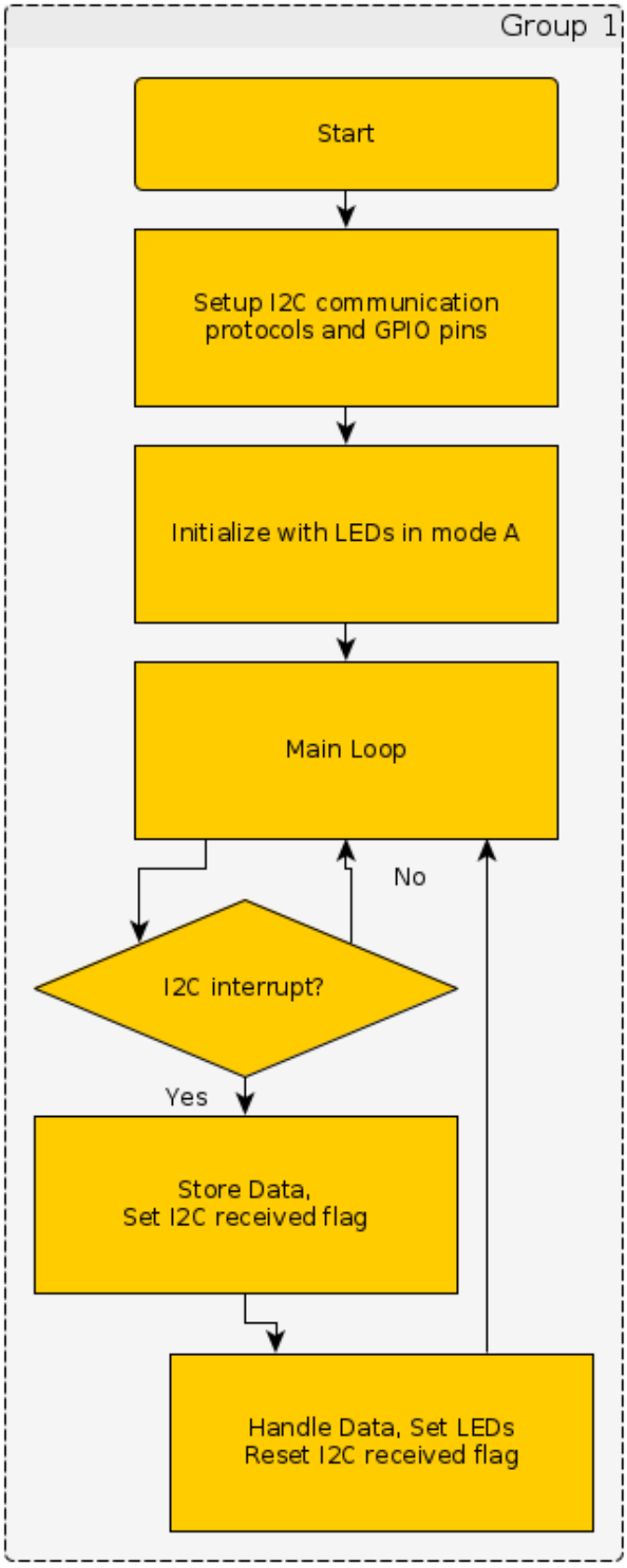


Figure 1: MSP430FR2310's code