# Project 3

**Keypad and LED Patterns**

EELE 465

**Due date: 02/25/2025**

---

**Acknowledgement** This handout is based off of previous handouts by Randy Larimer and Cory Mettler.

**Project Goal** Use a 16-button keypad to select which LED pattern to display on an LED bar.

In this project, a keypad will be used to select from different time-varying patterns displayed on eight LEDs using your MSP430 and a breadboard. Patterns will be selected based on which button is pressed on your keypad. An unlock code will be required to begin the display of the LED patterns. An RGB LED will be used a status indicator.

This project is less explicit than the previous project was about the steps you need to take. Instead, there is a list of requirements and specifications your project must meet. However, I still give some guidance and requirements on your development process. This is so I can show you a systematic development process that will make your lives easier in future projects. In future projects, I will expect you to follow good development and testing practices. You will be graded on your development process!

This project and all future projects will be programmed in C (or C++ if you'd like, but you won't get as much support from me; or some or all parts can be in assembly if you want).

**GitHub Classroom assignment** https://classroom.github.com/a/q35R5m2 The GitHub assignment is a group assignment. If you need to change your team from last time, let me know and I will get that all sorted out on GitHub.

## Outcomes

After this lab you should be able to:

- Poll a 16-button keypad and determine which button was pressed.
- Interface with an LED bar.
- Control an RGB LED using PWM.

## Project Requirements

Gabby ~~**Requirement 1** The system must have a heartbeat LED~~

Iker **Requirement 2** Keypad lock/unlock behavior

The system must start in a "locked" state, waiting for a 4-digit unlock code to be entered. Once the correct unlock code has been entered (you get to choose your unlock code), the system will allow users to use the keypad to control the LED patterns.

~~**Specification 1** System must start in a locked state upon power-on reset and soft reset~~

**Specification 2** Unlock code must be 4 digits long

**Specification 3** **Extra credit** (2 pt) The unlock code must be entered within 5 s.

**Specification 4** **Extra credit** (1 pt) Pressing the "D" key will lock the system.

Iker **Requirement 3** RGB LED status indicator

An RGB LED must be used as a status indicator

**Specification 1** The RGB LED must be set to #c43e1d when the system is locked.

**Specification 2** The RGB LED must be set to #c4921d when the system is being unlocked.

**Specification 3** The RGB LED must be set to #1da2c4 when the system is unlocked.

**Specification 4** **Extra credit** (1 pt) The RGB LED is set to a different user-defined color for each LED pattern.

Gabby **Requirement 4** Display LED patterns on an LED bar

LED patterns will be displayed using 8 out of 10 bars on an LED bar. The key corresponding to the pattern number is used to select that pattern.

**Specification 1** When the system is locked, all LEDs must be off.

**Specification 2** After the system is unlocked, all LEDs must be off until a user selects a pattern using the keypad.

**Specification 3** When the next selected pattern is the same as the current pattern, the pattern must restart.

**Specification 4** When the next selected pattern is a different pattern, the new pattern must start within 1 s.

**Specification 5** When a previously selected pattern is re-selected, the pattern must start where it left off.

**Requirement 4.1** LED pattern transition period

The transition period for each pattern is specified as a multiple of the base transition period. The base transition period can be changed via the keypad.

**Specification 1** The default base transition period must be 1.0 s upon power-on reset and soft reset.

**Specification 2** Pressing the "A" key will decrease the base transition period by 0.25 s.

**Specification 3** Pressing the "B" key will increase the base transition period by 0.25 s.

**Requirement 4.2** LED pattern 0 (static)

**Specification 1** LEDs display `10101010`. This pattern does not change.

**Requirement 4.3** LED pattern 1 (toggle)

**Specification 1** LEDs toggle as shown below.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| step 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| step 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

repeat...

**Specification 2** LEDs toggle every 1.0 “base transition period” seconds.

**Requirement 4.4** LED pattern 2 (up counter)

**Specification 1** LEDs count up starting from 0

**Specification 2** LEDs count every 0.5 “base transition period” seconds.

**Requirement 4.5** LED pattern 3 (in and out)

**Specification 1** LEDs move in the following pattern:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| step 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| step 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| step 2 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| step 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| step 4 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| step 5 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

repeat...

**Specification 2** Steps transition every 0.5 “base transition period” seconds.

**Requirement 4.6** Extra credit (0.25 pt) LED pattern 4 (down counter)

**Specification 1** LEDs count down starting from 255

**Specification 2** LEDs count every 0.25 “base transition period” seconds.

**Requirement 4.7** Extra credit (0.25 pt) LED pattern 5 (rotate one left)

**Specification 1** LEDs move in the following pattern:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| step 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| step 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| step 2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| step 3 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| step 4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| step 5 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| step 6 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| step 7 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

repeat...

**Specification 2** Steps transition every 1.5 “base transition period” seconds.

**Requirement 4.8** Extra credit (0.25 pt) LED pattern 6 (rotate 7 right)

**Specification 1** LEDs move in the following pattern:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| step 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| step 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| step 2 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| step 3 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| step 4 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| step 5 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| step 6 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| step 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

repeat...

**Specification 2** Steps transition every 0.5 "base transition period" seconds.

**Requirement 4.9** **Extra credit** (0.25 pt) LED pattern 7 (fill to the left)

**Specification 1** LEDs move in the following pattern:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| step 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| step 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| step 2 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| step 3 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| step 4 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| step 5 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| step 6 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| step 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

repeat...

**Specification 2** Steps transition every 1.0 "base transition period" seconds.

**Requirement 5** **(only for groups of 3)** System must cycle through colors on a second RGB LED

Using a second RGB LED, the LED must cycle through all possible hues when the system is unlocked.

Convert from hue-saturation-value (HSV) or hue-saturation-lightness (HSL) to RGB to cycle through the colors.

**Specification 1** LED is off when the system is locked.

**Specification 2** LED continuously cycles through all hues when the system is unlocked.

**Specification 3** Pressing "*" on the keypad increases the cycle rate.

**Specification 4** Pressing "#" on the keypad decreases the cycle rate.

## Development Process

Here's a high-level overview of how you should go about developing this project. Note that planning is first! Unordered lists indicate that things don't need to be done in a particular order.

1. Planning
    - Plan out your software architecture.
    - Complete the workload distribution form (in `docs/admin/workload-distribution`).
    - Create and assign tasks using GitHub Projects.
    - Write pseudocode or create flowcharts for the main components.
    - Create a preliminary circuit diagram
2. Develop code and test basic keypad scanning.
3. Develop other modules. Write test programs for each module.
    - Keypad unlock behavior.

- RGB LED module.
- LED bar hardware.
- LED pattern software.
- etc.

### Git Procedure

Moving forward, you'll be expected to follow the generic git procedure (located on Brightspace and in your project repositories (`docs/admin/procedures`)).

Here's a rough outline of the desired procedure:

1. Do your initial planning activities in a `planning` branch.
2. Once your initial planning is done, create a pull request for the `planning` branch. All members must approve the pull request.
3. For each module (e.g., keypad, RGB LED, LED patterns, pwm, etc.):
    1. Create a new branch.
    2. Do any initial planning (e.g., pseudocode, flowcharts, diagrams) that wasn't done earlier. Do this and commit this before writing code.
    3. Implement the module. Create a test project to test the module.
    4. Create a pull request. The pull request must include evidence of your module working (e.g., screenshots from the AD2 showing that your code works, or some other form of validation).
    5. Have your team member(s) perform a code review.
    6. Once your team member(s) have approved the pull request, merge it into `main`.

No development should happen in the `main` branch. Everything should be done in a separate branch and be review/approved by the team via pull requests. When you're working together on integration, this should still be done in a separate branch.

### Repository Organization

Here's a suggested folder organization. You are free to deviate from this, but having a completely unstructured or haphazard repository will result in losing points.

- `docs`
    - `docs/assets`: where images go
    - `docs/admin`: where administrative stuff from me will be
        - `docs/admin/procedures`: style guides and git procedures
        - `docs/admin/rubrics`: grading rubrics
    - `docs/planning`: where your planning documentation will go

- `src`: all source and header files except `main.c`
- `app`: the CCS project folder, including `main.c`
- `test`: folder for containing all test projects/code
  - `test/keypad`: test project for keypad scanning

This structure isn't the "one folder structure to rule them all". As projects grow, optimal folder structures will change. Since we won't have many modules at this point, putting them all in one directory is fine. Having 100 modules in one directory, however, would likely not be a useful organization.

## Planning

Before writing any code, you need to make a basic plan of what you're going to do. On the technical side, this includes sketching out a system architecture (e.g., what components and modules you will create/use), writing pseudocode and/or creating flowcharts, and sketching a circuit diagram. On the managerial side, you will need to divide the work between team members and fill out the workload distribution form.

Keep in mind that you won't design the correct architecture, functions, or algorithms from the beginning—you'll always refactor as you iteratively develop your code. However, thinking things through will help get you closer to a good solution and help you avoid writing spaghetti code. 🍝👷‍♀️

> You must create a `planning` branch and complete the workload distribution, architecture sketch, pseudocode, and preliminary circuit/block diagram before writing code. You will create a pull request for the `planning` branch, which needs to be approved by all team members before creating other branches for writing code. You will lose points if you don't follow this procedure!

### Workload Distribution

Fill out the workload distribution form located at `docs/planning/workload-distribution.md`. You may want to sketch out your software architecture before making final decisions on how to split the work.

### GitHub Projects

We're going to try using GitHub Projects to plan, divide, and track your work.

Create a new project in your repository. Use the Kanban template.

We'll figure this one out as we go 🙂. I just want to expose you to some "agile" task management tools.

Update the tasks and add new ones as you work throughout the project.

### Software Architecture

Create a high-level software architecture diagram. This might be something like a C4 component diagram. It can also be thought of as a block diagram showing what software components will be used (e.g., keypad, i2c driver, pwm, rgb led module, etc.) and how they interact with and relate to each other.

An example will be provided in the lecture notes on Brightspace.

### Pseudocode or Flowcharts

Before writing any code, you should write pseudocode to plan out what your code needs to do. Granted, if the code truly is trivial, you don't need to write pseudocode. However, when in doubt, err on the side of writing pseudocode.

For the initial `planning` branch, you don't need to write pseudocode for everything. Just write pseudocode or create a flowchart for the high-level operation of the main loop.

For each individual development branch, I expect you to write pseudocode before writing any non-trivial code.

Pseudocode should go in `docs/planning/`. Flowcharts should go in `docs/assets/`, but the images should be included in a markdown file in `docs/planning/`.

### Circuit Diagram

Create a circuit diagram for your system before connecting your hardware. For sufficiently complex projects, you may opt to create a block diagram in place of or in addition to the circuit diagram.

## Keypad Module

### Basic Scanning

All partners need to be involved in the keypad scanning development. It's important that every member understands how it works.

Before coding, use the AD2's power supply and logic analyzer to verify that the keypad is working and that you understand how to determine which button is pressed.

Create a test project for developing the keypad module.

Your keypad module should save which button was pressed so the main application can use that information.

### Unlock Code

Once the basic keypad scanning is done, create a test project to implement the unlock functionality.

The whole group does not need to work on this.

## RGB LED Module

You should verify that your RGB LED is hooked up correctly and works *before writing any software*.

### Hardware

Unfortunately, the RGB LEDs are cheap ones from Amazon, so there isn't a datasheet. You'll need to figure out which pins correspond to what.

Calculate your current limiting resistors for each color. You may wish to measure the forward voltage for each color with a multimeter (I can supply one if needed).

Hook each color up to $V_{CC}$ to verify that everything works.

You should *not* directly hook the LED up to $3.3\,\mathrm{V}$ without current limiting resistors. Some of the colors may work fine at $3.3\,\mathrm{V}$, but you run a risk of burning out the colors that have lower forward voltages (trust me, I've done it many times, and will continue to do so...)

### Software Control

Implement a module that can set the value of the red, green, and blue channels. Create a test project for this.

Think pulse-width modulation (PWM) and timer modules.

## LED Patterns Module

You should verify that your LED bar hardware works correctly *before writing any software*.

### Hardware

1. Review the datasheets for the LED bar and the resistor arrays. If you are unsure why you need a resistor array, please ask before blowing anything up. 💥
2. Connect the parts on your breadboard and manually test the LEDs by jumping $V_{CC}$ to each LED.
3. Write a simple test program that lights up some or all of the LEDs using GPIO pins.

### Software Patterns

Create a test project that implements all the LED patterns you need. You can hardcode which pattern gets run, you can cycle through the patterns, use a button to change patterns—whatever you want, just keep it simple (it's just a test).

# Demo

1. Introduction
    - Introduce the project at a high level. Don't assume the instructor or TA knows about the project. Focus on the goal of the project.
    - Use a high-level flowchart of the main program loop to help describe the system's functionality.
    - Use a circuit diagram to explain how the circuit was constructed.
    - Use a software architecture diagram to discuss how the software was *designed*.
2. Demonstration
    - Demonstrate the full system working. You may choose the order in which you would like to present functionality.
    - Be sure to demonstrate that every specification was met.
    - The partner that was primarily responsible for a subsystem/requirement/specification should present that functionality.
    - If you completed any extra credit, demonstrate that functionality.
3. Review of results
    - The instructor or TA will review the final workload distribution form with the group. Be prepared to discuss any discrepancies between the workload distribution form and how the work was actually divided. The workload distribution form may be updated as necessary.
    - The instructor or TA will review what percentage was achieved for each specification and the resulting points for each partner.