UF UNIVERSITY of FLORIDA
Herbert Wertheim College of Engineering

Computer Engineering
CEN3907C

Drivers for Embedded Systems

Overview

Embedded systems are all around us, in our homes, vehicles, toys, and workplaces. As Computer Engineers with a focus on the nexus of hardware and software, embedded devices serve as a playground to experiment with hardware-level coding. In this lab, you will explore embedded development by creating two drivers and an example application for the hardware kit you have been provided.

Learning Objectives

- Learn how to write C/C++ code for the Raspberry Pi RP2040.
    - Understand the architecture and interfaces of the **Pico SDK**.
    - Understand the **CMake Configuration System** used by the Pico SDK.
- Create well-engineered drivers that hide the low-level implementation details of hardware devices. These drivers will be delivered as libraries that present the application software with intuitive, safe, and robust interfaces.
    - Write an **I2C driver** for the LIS3DH accelerometer.
    - Write a **NeoMatrix** display driver.

Resources

The following resources, in addition to the tips and hints below, should provide you with enough information to complete this assignment. If you have not done so already, follow the instructions in the HW Intro document to compile and load an example from *pico-examples* onto your board.

- Getting Started with Raspberry Pi Pico
    - Provides detailed guidance on building and running examples, as well as instruction on how to create your own projects (Chapter 8).
- RP2040 Datasheet
    - Provides comprehensive coverage of the RP2040 chip. An especially useful resource for this assignment is the GPIO-Peripheral mappings in section 2.19.2.
- Pico SDK Manual
    - Provides a comprehensive overview of the build system and interface libraries available in the SDK.
- Pico SDK Repository
    - In order to use the interfaces provided by the SDK, you must clone this repository to your development computer. It can be located anywhere that *cmake* can find it – but it is recommended that you clone it to a directory where you keep your RP2040 C/C++

projects. *cmake* will find the SDK via the PICO_SDK_PATH environment variable, which can be an absolute or relative path.

- Pico SDK Examples Repository
  - This repository contains buildable code examples of using the SDK libraries for various functionality.
- LIS3DH Datasheet
  - Provides register mappings and definitions for the accelerometer.

Specification

For this lab, you shall implement the following two drivers as C++ classes, as well as the example application specified below. Both classes should have header files and C++ files for use by the application software.

Accelerometer Driver

The Propmaker FeatherWing includes an **LIS3DH** accelerometer connected to Feather I2C pins. Your class should be called LIS3DH and implement the following components:

```
float x, y, z;
```
Class members representing the three acceleration values in units of g (Earth's gravitational acceleration).

```
LIS3DH(void);
```
Class constructor. Used to instantiate an object.

```
bool init(void);
```
Initializes the accelerometer, returning true on success or false on failure.

```
void set_reg(uint8_t reg, uint8_t val);
```
Set a register on the LIS3DH to the given value.

```
uint8_t read_reg(uint8_t reg);
```
Reads and returns the byte at address *reg* on the accelerometer.

```
void update(void);
```
Updates the class members x, y, and z with the current acceleration values.

Tip: the SCL & SDA pins on the Feather RP2040 are not the same as the default pins on the Pico. You can find the pinout by looking at the schematic. The pins used by the Feather are only usable with one of the two I2C instances on the chip – find out which one by looking at the GPIO Functions table in the RP2040 Datasheet.

Hint: All of the necessary functions for configuring and reading data from the LIS3DH are available in the *pico-examples* repository. Your goal is to (slightly) adapt them, and integrate them into your class, rather than being called from the application code.

NeoMatrix Driver

- The **NeoMatrix** included with your hardware kit is an array of 64 WS2812 RGB LEDs. These addressable LEDs are driven by a one-wire, asynchronous protocol. Your class should be called NeoMatrix and implement the following components:

  ```
  NeoMatrix(uint8_t width, uint8_t height);
  ```
  Class constructor. Used to instantiate an object.

  ```
  bool init(void);
  ```
  Initialize the object, returning *true* on success or *false* on failure.

  ```
  void set_pixel(uint8_t row, uint8_t col, uint32_t color);
  ```
  Set the pixel at row *row* and column *col* (zero indexed) to *color*.

  ```
  void clear_pixels(void);
  ```
  Set all elements of the pixel buffer to 0x00.

  ```
  void write(void);
  ```
  Write the pixel buffer to the NeoMatrix.
- The driver should be based on a PIO implementation of the WS2812 protocol. See the example in the *pico-examples* repository for a starting point. Your *CMakeLists.txt* file is where you define the compilation of the PIO program, and the functions that are defined in the resulting header file can be used in your driver class.

Hint: the *ws2812.pio* file in the *pico-examples* repository can be directly copied to your project. You might consider renaming the program and the static functions below it to your preference; however, the content of the pio assembly should not change. Once you compile the pio header file (using your *CMakeLists.txt* file), you can *#include* that in your driver class and call the relevant functions to setup and write data to the state machine.

Hint: the Propmaker FeatherWing has a pin to enable the power to the connected NeoPixels. This pin is pulled low (disabled) by default, so in addition to setting up the PIO peripheral, you must drive this pin high using the SDK GPIO configuration functions as seen in the *blink.c* example.

Hint: the initialization function in the *pio/ws2812* example contain a boolean parameter as their last argument. This tells the state machine whether the LEDs expect 24 bits of color information (RGB) or 32 bits (RGBW). The LEDs on your NeoMatrix expect only 24 bits, so make sure this parameter is *false* when calling the initialization function from your driver.

Example Application

- A simple **Bubble Level** example application should be included in your code. This demo app should turn your device into a level where the position of the device indicates if it is level or not.
- The code for this app should be called *level.cpp* and be in the *src* directory.
- When the x and y acceleration values are within +-0.1g, the center 4 LEDs on the matrix should light up solid green.
- Otherwise, only one LED should be illuminated in red in a location that indicates the axis of tilt.
- After invoking **cmake** and **make**, the build system should create a *level.uf2* file in the *build* directory that can be loaded onto the Feather RP2040

Hint: the *CMakeLists.txt* file can be tricky to understand – chapter 8 of the Getting Started document is the best place to begin. Following this guide, try first getting a bare bones template working with a single source file (perhaps printing something over USB serial). In order to get *printf* and *puts* functions to print to the USB serial port, you must include the *cmake* function *pico_enable_stdio_usb(<your-executable> 1).* Once that is working, start incorporating your libraries.

Submission

Your submission will be composed of the following elements:

- Your complete source code as a **TGZ archive** named *embedded_driver.tgz*. This archive will be unpacked, and your submission will be built using CMake and Make in Gradescope upon submission.
- **Report in PDF format (see below).**

TGZ Archive

Your TGZ archive (*embedded_driver.tgz*) should have the following directory/file structure:

```
embedded_driver
├── src
│   └── *.cpp
├── ...
├── CMakeLists.txt
├── pico_sdk_import.cmake
└── README.md
```

- **Notice:** The archive contains a directory named *embedded_driver*. Your archive **must** also do so.
- The README.md file should be personalized and have specific explanations about your project's structure, as well as instructions on how to build and load your code.
- Your *CMakeLists.txt* file should include all information necessary to build and link your program, as well as create the loadable *level.uf*2.
- You will need a *build* directory for the build system to put output files in; however, this directory should not be included in your submission, as we will create it during grading.
- The *.gitignore* file is not required, but it is okay if it is present.

Report

When filling in the report template, please include responses to the following:

- What challenges did you encounter when implementing your driver libraries? How did you overcome them?
- Do you have any suggestions for improvement to the structure of these libraries?
- Please include one oscilloscope screen capture for the I2C bus and NeoMatrix driver line, with a brief explanation of their operation.