Last modification: December 27, 2021

NSF Workshop
“Hello, World!”

1 Introduction

In this exercise, you’ll be introduced to programming a microcontroller. We’ll provide you with some skeleton code (i.e., code that is partially written but still needs some meat on its bones) and ask you to change parameters and logical elements to achieve the desired behavior.

Though the end result will hardly be groundbreaking – you’ll simply turn an LED on and off – it will demonstrate basic functionality and prove that your setup is alive and well. Starting with the basic program provided in this activity, you will add functionality throughout the course until your robot is able to complete the complex tasks for the final project.

Unlike the trivial `Blink` example provided by the Arduino IDE, you will implement similar behavior using a *state machine*, which will serve as the framework for almost everything you do with your robot this term (and many future projects). A state machine is an important tool that can be used to manage complex behaviors in robots, and you will be expected to develop and implement more of the details as the course progresses.

1.1 Objectives

Upon successful completion of this activity, a student will be able to:

- Edit and upload code to a microcontroller, in this case the microcontroller on the Romi Control Board,
- Use basic library routines to read inputs and control outputs,
- Use program flow control (`for`, `while`, and `if..else` statements) to control the behavior of a microcontroller, and
- Describe how a simple state machine can be used to control a robot’s behavior.

The specific board that you’ll use is the Romi Control Board, which we’ll often refer to simply as your “Romi”. The “brains” of the Romi is the `ATmega32u4`, a low-cost microcontroller from Microchip that is the same microprocessor found on the Arduino Leonardo. The Romi Control Board includes the chip, several sensors, and a number of useful interfaces: a USB connection for programming, pins for connecting additional sensors and circuits, and so forth. Figure 1 shows an annotated diagram of the Control Board.

2 Resources and preparation

To get started,

- Read How do microcontrollers work?
- Install an Integrated Development Environment (IDE), which you will need for programming. For this course, we highly recommend `VSCode` + the `platformio` extension, which is what all of the example codes have been developed for.

  1. To install both `VSCode` and `platformio`, follow the instructions on the platformio webpage.
  2. Read through the Quick Start Guide for how to set up a project.
  3. Familiarize yourself with the PlatformIO Toolbar.

  `platformio` is a flexible Integrated Development Environment (IDE) that works on all the common operating systems. The setup and workflow is a little different than with the Arduino IDE, but `platformio` has some nice features, such as automatic code completion, the ability to click into function definitions, and `github` integration. In short, it is a more professional development environment than the Arduino IDE, so we prefer it for code development. We hope you will spend some time exploring its capabilities.

  Concerned about privacy? By default, both `VSCode` and `platformio` share telemetry with developers. Instructions for disabling telemetry can be found on these pages:

  - `VSCode`
  - `platformio`

- Obtain the example codes, which are provided through a `github` *repository* on a github site. You have two options for obtaining example codes:

  1. **Clone a repository using Github Desktop**. First, install Github Desktop. After installation, use your web browser to navigate the workshop sample code page. Click the green "Code" button and choose "Open with Github Desktop". Follow the instructions and put the repository in the folder of your choice.
  2. **Download the examples as a `.zip` file and unzip it.** Use your browser to navigate the workshop sample code page. Click the green "Code" button and choose "Download ZIP". Save the `.zip` file and unzip it into the directory of your choice.

- If you have not done so already, assemble your Romi. Instructions for assembly your Romi can be found in the Assembly Guide.

2.1 Materials

- A Romi with Pololu's Romi Control Board.
- A USB cable to connect to your compuuter.

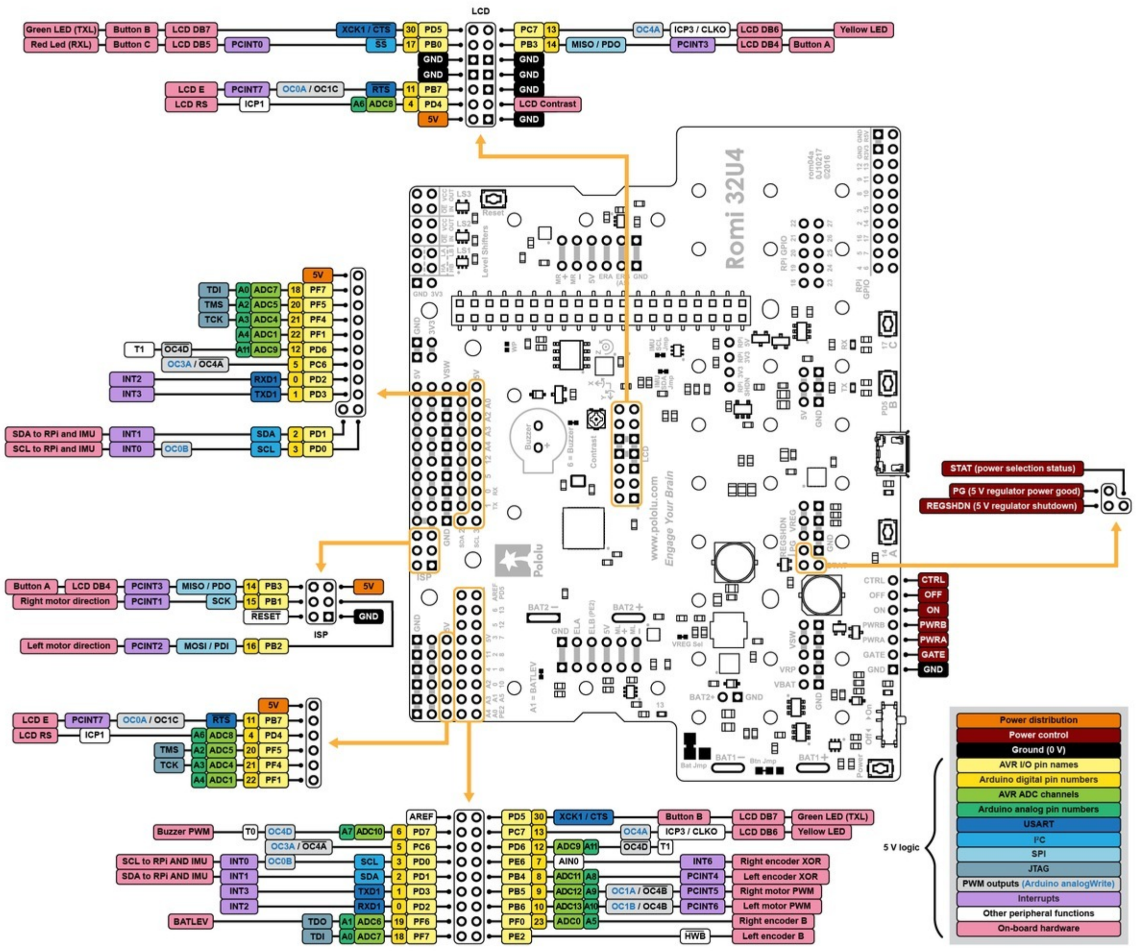

Figure 1: Pinout of the Romi Control Board. From Pololu’s Romi User Guide.

Activities

3 Overview

Your goal is to program your Romi such that when you press one of the buttons on the Control Board, an LED will toggle on and off. You will also need to add code to keep track of how many times the LED was toggled.

3.1 Functionality

A major theme of this class will be robot *functionality*. Functionality refers to the *tasks the robot has to do*, regardless of how it actually accomplishes those tasks. Of course, the physical components that make up the robot are essential to how it performs all of the necessary tasks, but by focusing on functionality, you will learn how to approach problems in robotics beyond the specific problems presented in this class.

Typically, we’ll approach each problem from the perspective of “Sense. Think. Act.” In that way, we’ll highlight the integrated nature of Robotics – how functionality often relies on concepts from multiple disciplines and how those concepts are put together to make a robot.

4 I/O Fundamentals

Physically, input and output with a microcontroller happen through *pins*. Some of the pins on the microcontroller chip itself are connected to the larger pins on the edge of the development board, which allow you to easily connect other circuits, for example with the use of a breadboard. Each of the pins is labelled (e.g., `A0` or `TX` or `GND`). Some of the pins have special functionality, and we’ll explain some of that functionality throughout the course.

4.1 Digital input: Reading a button

For your microcontroller to sense anything, signals must be *input* into the microcontroller. Common examples include reading a button or, as you’ll see in a later activity, reading an ultrasonic rangefinder. Here, you’ll use the microcontroller to respond to presses of one of the built-in buttons on the Romi Control Board.

Your Romi Control Board has four buttons built into it. One is a reset button, labelled `Reset`, that will restart your program when you press and release it. The others are labelled `A`, `B`, and `C`, and can be used to register user input on pins `14`, `30`, and `17`, respectively (see Figure 1). Note that these pins can also be used for general input and output by connecting wires to the appropriate socket on the Romi Control Board. You should avoid using the buttons when you do so.

Procedure

1. Connect the Romi Control Board to your computer with a USB cable.
2. Open VSCode.
3. Click on `File -> Open Folder...` and select the folder titled `activity00.button`.
4. Click the Compile button at the bottom of the IDE (the checkmark). If all goes well, it’ll say "Success" in green. If not, see the Troubleshooting Guide below for pointers or contact the workshop staff.
5. Connect your Romi to your computer with a USB cable. Press the Upload button (the right-pointing arrow at the bottom of the window). Again, you should see “Success” in green.

4.2 The Serial Monitor

Debugging an Arduino program is not as easy as debugging a regular computer program. Without special hardware, you can’t stop the program and probe values in the program itself. Instead, you’ll often make use of the Serial Monitor, which is a convenient way to send information from your board to your computer (and vice versa). That way, you can output notifications or important values to help you understand what's going on (or what's going wrong) with your robot.

To use the Serial Monitor, you must call,

```
  Serial.begin(<baud rate>);
```

in the `setup()` routine, where `<baud rate>` is the speed that the microcontroller will send data, in bits per second. Common choices are 9600 (the default in "Arduino") and 115200 (a recommended higher speed). In the sample programs, the baud rate will be `115200`, so *you need to tell platformio to use the same baud rate*, otherwise you'll get gibberish. This is done with the line in the `platformio.ini` file that reads:

```
monitor_speed = 115200
```

In your code, you tell your robot to write to the Serial Monitor using,

```
  Serial.print(<data>);
```

and

```
  Serial.println(<data>);
```

where the second version will send a newline character after it sends the data, causing the output to scroll. `data` can be text or numbers, where the former must be enclosed in quotes, e.g.:

```
  float temperatureA = 22.5;
  float temperatureB = 30.1;

  Serial.print(''Temperature A = '');
  Serial.println(temperatureA);
  Serial.print(''Temperature B = '');
  Serial.println(temperatureB);
```

would output:

```
  Temperature A = 22.5
  Temperature B = 30.1
```

The `float` denotes that a variable can have decimal places – in computer-speak, it's referred to as a *floating-point* number.

Procedure

1. Open the Serial Monitor (see the `platformio` Quick Start for instructions). Press the button labelled `A` on the Romi Control Board. You should see a response in the Serial Monitor every time you press the button.

Later, you'll make edits to the code, but first, let's discuss how the code works. The first few lines declare some variables to manage the hardware. The most important thing to note is the creation of a variable, `robotState`, that will allow the program to keep track of what it is doing. In this simple example, the robot is either active or idle (which are really just placeholders for now – we'll fill in more actions and more states as the term progresses).

The `setup()` function, which is run once each time the program restarts, does some more initialization. Then we get to the `loop()` function, which is where the bulk of the work happens. If you've worked with Arduino before, the `loop()` might look a little different than you're used to. In this program, we use a *state machine*, which is overkill for something as simple as turning an LED on and off, but as we progress through the course, using a state machine will simplify the program as a whole.

Note the following:

- The physical button is read using a `Romi32U4ButtonA` object. We'll discuss more about object-oriented programming later. For now, just find the line of code that detects when the button is pressed.
- Physically, when the button is up, the terminals on the button itself are not connected electrically. A special resistor, called a *pullup resistor*, is used to guarantee that the pin will read `HIGH` when it's not pressed. When the button is pushed down, a circuit is closed and the pin on the microcontroller is connected electrically to ground so that the pin will read `LOW`.
- The program repeatedly *polls* the pin connected to the button to determine if the voltage has changed. That is, the library doesn't just look to see what the voltage is, but if the voltage has *changed* from the last time it checked.

4.3 Digital output: Lighting an LED

While getting information into your robot – sensing – is critical to its operation, we also want our robots to take actions. To do that, you'll need the microcontroller to output different signals. Here, we'll explore *digital output*, where the microcontroller will output one of two voltages: `HIGH` or `LOW`. Quite a lot can be done with digital output, but here we'll just turn an LED on and off.

In this activity, we use one of the on-board LEDs built into the Romi Control Board. Optionally, students can build a simple LED circuit on a breadboard to give them an introduction to basic electronics.

Procedure

1. Define the LED pin as a constant by adding,

```
#define LED_PIN 13
```

near the top of the code. This will tell the compiler to substitute “`13`” for any instance of `LED_PIN` in your code (much like “Find and Replace” in a text document, but at compile-time). Doing so will make it so you only have to change one value if you want to change the LED pin – a much easier way to manage your code than changing the numbers everywhere.

2. In the `setup()` routine, add a call to `pinMode()` to declare the LED pin to be an output.

Solution: Most students have used Arduino, but those that haven’t will need to look up the syntax,

```
pinMode(LED_PIN, OUTPUT);
```

3. In the state machine in the `loop()`, add calls to `digitalWrite()` to turn the LED on or off, as appropriate.

Solution: Most students have used Arduino, but those that haven’t will need to look up the syntax,

```
digitalWrite(LED_PIN, HIGH);
```

and,

```
digitalWrite(LED_PIN, LOW);
```

4. Upload your new code. Does the LED go on and off when you press the button?

5. Add a variable, `count`, to the code and initialize it to 0. Add code to add 1 to `count` and print the current value every time the LED turns on.

Solution:
Define a variable,

```
int count = 0;
```

In the `handleButtonPress()` function, add the line

```
count++;
```

to the section that switches from IDLE to ACTIVE

6. Test your system by pressing the button 10 times to verify that each press was registered.

5 Troubleshooting Guide

There are always a few things that go wrong with any project. Here, we list a few of the common issues that we see with `platformio` and the Romi. You are always welcome to contact the course staff if you are having troubles.

- **Compile tools unavailable**. You've opened a folder in VSCode, but it can't find `platformio` or there's no compile button. You have to open folders "at the right level". A permittable folder corresponds to one project code. If you open the `activities` folder instead of `activities00.button`, `platformio` will get confused because the former doesn't correspond to a single project. Any folder you open must contain at a minimum a `src` folder and a `platformio.ini` file.
- **Uploading repeatedly fails**. The code compiles, but you can't upload – `platformio` says "looking for upload port," but it can't find one. This occurs when the microcontroller gets in a state where the USB is tied up (formally, it can't execute and interrupt to read from the USB stream). This problem is almost always fixed with a "double-tap" of the reset button, which forces the microcontroller into *bootloader mode*. Timing is important, but folowing these guidelines generally works:
  - Click the upload button and then immediately press the `Reset` button twice in rapid succession. Let the computer try to upload. If it works, you're done. If not,
  - Do the procedure again, but this time, as soon as the upload tool says that it's looking for an upload port, double-tap again.

  Again, timing is important, so you may need to try a few times.