Lab 2: External Actuators for Arduino with CircuitPython

A. Trahan, P.E.

September 19, 2019

Introduction

This lab provides an overview of using external actuators (output devices) on Arduino with Circuit Python. While there are many built-in actuators on the CPX, there may be additional actuators (like servos) that need to be attached for a given project. In addition, most other Arduino devices don’t have as many built-in actuators, so all actuators for a project must be attached externally.

The CPX has a variety of ”pins” for inputs and outputs. Instead of tra ditional pins, the CPX uses alligator clip mounts around the edge of the device. Since these are mapped to pins on the SAMD21 chip on the CPX (the ”brain”), and they’re actually pins on some Arduino devices, so refer ences will often call them pins. The clip mounts on this device are generally called pads, so we will use that nomenclature. Some pin definitions from the SAMD21 are connected to on-board sensors and actuators, but this lab will focus on the externally accessible pads.

External sensors (inputs) and actuators (outputs) are driven directly by controlling the voltages applied to these pads. This is the only way to in terface with custom elements, but for many common elements (e.g. servos), there are libraries that make it eaiser to write code using their functionality.

In this lab we will:

1. Connect directly to CPX pads/pins

2. Add libraries to Circuit Python on the CPX

1

3. Use external actuators to respond to sensors on the CPX Required materials (per group):

*•* (1) Circuit Playground Express (CPX)

*•* (1) Micro USB cable

*•* (1) Breadboard

*•* (1) LED (any color)

*•* (1) Piezometric buzzer

*•* (1) Servo (SG90)

*•* (1) Resistor [1 *k*Ω]

*•* (1) Resistor [100 Ω]

*•* Assorted jumpers and alligator clips

1 Connecting to Pads/Pins on the CPX

The CPX has fourteen pads along the edge - six power pads for power/ground and eight I/O pads for signals (Figure 1). A full definition of the pads can be found at the Adafruit website, but a few to note:

1. GND - There are three ground pads

2. 3.3V - There are two 3.3V power pads

3. Vout - This pad carries the (higher of) battery or USB voltage, with a 1 Amp fuse

4. A0 - This is a true analog pin, so it can carry real waves (inc. audio)

5. A1-A7 - There are three digital I/O pads, many with pulse-width modulation (PWM) and capcitive touch options

As an example, try connecting an LED to pad A1:

2

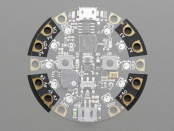
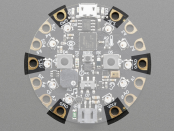
(a) Power Pads (b) I/O Pads

Figure 1: Pads on the CPX for connecting to external sensors and actuators

1. Follow the circuit diagram in 2 to connect an LED to pad A1

(a) Use an alligator clip jumper to connect A1 to a breadboard row

(b) Use an alligator clip jumper to connect GND to the breadboard ground column

(c) Connect an LED to the breadboard with positive (long) lead in the same row as the A1 jumper

(d) Connect a 10k ohm resistor to the breadboard ground column and the row with the LED negative (short) lead

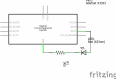
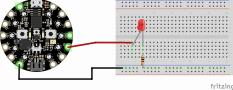
(a) Breadboard layout (b) Circuit diagram

Figure 2: Diagrams for attaching external LED to CPX

2. Enter the code below in Mu, and save to the CPX as code.py. A few things to note:

3

(a) We import board to map python commands to pins/pads

(b) We import digitalio to control signals being passed through the pins/pads

(c) To access the pin/pad, create a DigitalInOut object connected to it, then assign it a direction (in/out)

(d) This example is available online: Adafruit Digital Outputs Tuto rial

3. This can also be controlled live via the REPL console by entering the Serial console and hitting Ctrl-C (Cancel). Try entering the command: led.value = not led.value

*#-- ExternalLedBlink.py - CircuitPython code for CPX --#*

*# Import Section*

import board

import digitalio

from time import sleep

*# Setup Section*

led = digitalio.DigitalInOut(board.A1)

led.direction = digitalio.Direction.OUTPUT

*# Loop Section*

while True:

led.value = True

sleep(0.5)

led.value = False

sleep(0.5)

2 Adding Libraries for External Actuators

Directly modifying pins provides the most granular control for the CPX, but many actuators and sensors have packages mapping pin control to function ality. For example, in Lab 1, set neopixel brightness with a single command,

4

even though there are actually several pins involved. Actuator and sensor libraries allow us to focus on coding for functionality and often make code faster to write and easier to read.

As an example, we will install the Circuit Python Library Bundle. Then we will walk through an example using an external buzzer via direct pin control vs. via an I/O library. The following examples were developed using the Adafruit Piezo Buzzer Tutorial.

2.1 Installing the Circuit Python Library Bundle

1. Each version of Circuit Python requires a different bundle. Begin by checking your Circuit Python version

(a) Open Mu and connect the CPX

(b) Enter the REPL console by opening the Serial console and hitting Crtl-C

(c) This will automatically print the current version above the first input line. Note the major version number. As an example, the CPX in Figure 3 is running CircuitPython version 3.

Figure 3: Finding the CircuitPython version with Mu

2. Download the latest version of the bundle for your version from https: //circuitpython.org/libraries and unzip the bundle.

3. To add a library to the CPX, copy it from the lib directory in the bundle into the lib directory on the CPX (create one if it does not exist)

4. For the buzzer exercise, we will be using SimpleIO.mpy, so copy that into the CPX lib directory

5

2.2 Using a Buzzer via Direct Pin Control

As a first example, we will run the buzzer using direct pin control. A piezo electric buzzer has an element that deforms when subjected to a voltage. By turning that voltage on and off, the element oscillates at a chosen frequency, generating a sound. This can be done with a true analog signal (e.g. a sine wave), but most of the pins on an Arduino are digital - they can be on or off. To get around this, many some pins are attached to an oscillator on the device and provide ”pulse width modification” (PWM), which turns them on and off at a chosen frequency, emulating an analog signal. The pulsio package allows us to control PWM signals over pins.

You can learn more about PWM in the Adafruit PWM Outputs Tutorial. Try connecting and using a buzzer as an example of PWM control:

1. Connect the buzzer to the CPX. Wiring is very similar to the LED example above, simply change the actuator and the resistor. Lower resistance gives higher volume, so try a 100 ohm resistor. See Figure 4.

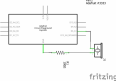
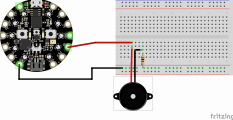
(a) Breadboard layout (b) Circuit diagram

Figure 4: Diagrams for attaching external Buzzer to CPX

2. Enter the code below in Mu, and save to the CPX as code.py, and note the following:

(a) We import pulseio to control PWM signals being passed through the pins/pads

(b) To access the pin/pad, create a PWMOut object with variable fre quency connected to it

6

(c) The buzzer frequency determines the tone it will create

(d) The buzzer duty cycle determines what percent of time power will be on and off. Setting this to 50% maximizes the on/off flipping, which is what drives the buzzer. The duty cycle variable is a 16-bit integer, so half power is approximately 215.

*#-- ExternalBuzzerDirect.py - CircuitPython code for CPX --#*

*# Import Section*

import board

import pulseio

from time import sleep

*# Setup Section*

buzzer = pulseio.PWMOut(board.A1, variable\_frequency=True) buzzer.frequency = 440 *# This is an A3*

OFF = 0

ON = 2\*\*15

*# Loop Section*

while True:

buzzer.duty\_cycle = ON

sleep(1.0)

buzzer.duty\_cycle = OFF

sleep(2.0)

2.3 Using a Buzzer via SimpleIO

The same can be achieved in fewer lines of code using an external library. This is a relatively simple example, so using the library doesn’t change much, but for more complicated actuators external libraries can save a lot of time and greatly improve the readability of your code. To drive the buzzer with the simpleio library:

1. Be sure the library is installed. The file SimpleIO.mpy should be in the CPX lib directory, as described above.

7

2. Use the same connections as the previous example (Figure 4)

3. Enter the code below in Mu, and save to the CPX as code.py, and note the following:

(a) We import simpleio to access the library

(b) simpleio has a tone function that plays a given frequency through a given pin/pad for a given duration

*#-- ExternalBuzzerLibrary.py - CircuitPython code for CPX --#*

*# Import Section*

import board

import simpleio

from time import sleep

*# Setup Section*

freq = 440 *# This is an A3*

*# Loop Section*

while True:

simpleio.tone(board.A1, frequency=freq, duration=1.0) sleep(2.0)

3 Responding to Sensors on the CPX

Previous examples have run external actuators on schedules set in the code, but often systems will need to respond to sensors. As an example, we will use a servo as a needle gauge to show how bright a room is, as read by the CPX’s on-board light detector (Figure 5).

Steps for this example:

1. Install the adafruit motor library by copying it from the bundle to the CPX lib directory. Note that this is a directory, not just a single file.

8



Figure 5: Light sensor on the CPX

2. Connect the servo to the CPX as shown in Figure 6. With so few parts, this can even be done without a breadboard.

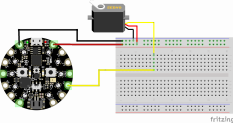
(a) Breadboard layout (b) Circuit diagram

Figure 6: Diagrams for attaching external Servo to CPX

3. Enter the code below in Mu, and save to the CPX as code.py, and note the following:

(a) We first need to define a PWM pin with the correct frequency to control the servo (as given in specs).

(b) We can then attach a servo object to that pin. The min and max pulse arguments define the range of the device and may vary

9

depending on your servo (especially for inexpensive models). If the range of motion seems incorrect, try adjusting these.

(c) There’s a new Section, the Function Section, in which we define the conversion between light sensor reading (0 to 320) and the servo position (0 to 180 degrees).

(d) To set the servo position, we read the light sensor (cpx.light), map that to a servo position via the function, then set the angle parameter of the servo object.

*#-- ServoLightGauge.py - CircuitPython code for CPX --#*

*# Import Section*

import board

import pulseio

from adafruit\_circuitplayground.express import cpx

import adafruit\_motor.servo

from time import sleep

*# Setup Section*

pwm = pulseio.PWMOut(board.A1, frequency=50)

servo = adafruit\_motor.servo.Servo(pwm, min\_pulse=750, max\_pulse=2600)

*# Function Section*

def light\_to\_servo\_pos(light):

light\_max = 320

return 180 - ((light/light\_max) \* 180)

*# Loop Section*

while True:

servo.angle = light\_to\_servo\_pos(cpx.light)

print(servo.angle)

sleep(0.5)

Additional ideas and questions:

10

1. How would you change the direction of the gauge? Try programming it such that the needle swings the other direction for light.

2. After the last lab, you’re quite familiar with the neopixels. Try pro gramming the neopixels to light up (number or brightness) as the light in the room increases - this is like the automatic brightness response on a cell phone.

3. Extreme brightness can be harmful to the eyes. Try adding a buzzer that chirps when the surroundings are too bright.

4. The CPX has many more sensors (sound, accelleration, etc.). Pick one and make your actuators respond to it instead of the light sensor. Remember the API for the cpx module can be found here: Circuit Python cpx Module API.

11