Experiment 1: Digital Input and Output

In the REPL section, we saw how to control an LED by individually calling commands in the REPL. This time, we are going to show two examples:

* How to respond to a button press
* How to blink the onboard LED by writing code in a text editor and uploading that file to the ESP32.

There is one minor problem with MicroPython on the ESP32 that we will address in this experiment: if your code includes a forever loop, the MicroPython interpreter will continue to run your code (forever!), which means we never get a REPL prompt.

The only way to stop a forever loop like this is to open a serial connection to the ESP32 and send it a *ctrl* + *c* command to terminate the running program.

The ampy tool requires REPL to upload new programs, so if the ESP32 is stuck running one of our programs, you will not be able to upload a new program. In the following experiment, we will build a tool that allows us to press the ESP32 Thing's onboard button (labeled *0* on the board) to stop our program and return to REPL.

Hardware Connections

Code Part 1: Push Button, Get REPL

In your text editor of choice, enter the following code:

COPY CODE**import** machine

**import** sys

**import** utime

# Pin definitions

repl\_button **=** machine.Pin(0, machine.Pin.IN, machine.Pin.PULL\_UP)

# Wait for button 0 to be pressed, and then exit

**while** True:

# If button 0 is pressed, drop to REPL

**if** repl\_button.value() **==** 0:

print("Dropping to REPL")

sys.**exit**()

# Do nothing

utime.sleep\_ms(1)

Save the code with a name like *button.py*.

**Run It:**

To upload the program to the ESP32, we first need to rename it to *main.py*. In most configurations of MicroPython, you will find two files stored on the microcontroller: *boot.py* and *main.py*. As soon as the microcontroller receives power (or directly after a reset), the code in *boot.py* will be run. Then, *main.py* will run. Note that once *main.py* finished executing, it will be run again (over and over again forever).

In essence, *main.py* acts as a forever loop for our code. However, rather than trying to deal with uploading 2 files to the ESP32, we are just going to worry about putting our code in *main.py* and uploading that.

Open a command terminal on your host computer (not a serial terminal!). Navigate to wherever you stored your *button.py* file. For example (on Windows):

COPY CODEcd Documents\Projects\Python\MicroPython

Then, enter the following two commands to create a copy of your code named *main.py* and upload it to the ESP32 (change <PORT> to the serial port of your ESP32, such as COM7 for Windows, /dev/cu.usbserial-<some letters/numbers> for macOS, or /dev/ttyUSB0 for Linux):

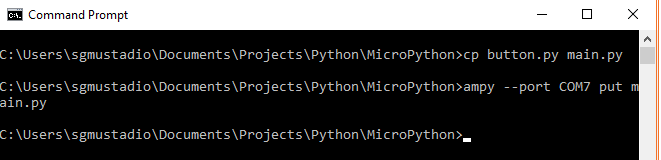
COPY CODEcp button.py main.py

ampy --port **<**PORT**>** put main.py

If you are on Windows and your port number is higher than 10, ampy may give you an error message like so: ampy.pyboard.PyboardError: failed to access \\.\COM24. There is a known issue with Python's recognition of Windows ports above 10, but ampy does not take this into account, so we must format our port explicity for Python. Try formatting your COM port as you see it here (replacing <PORT> with your port number/location): [\\.\<PORT](file:///\\.\%3cPORT)>

**Note:** If you are on Windows and your port number is higher than 10, ampy may give you an error message like so: ampy.pyboard.PyboardError: failed to access \\.\COM24. There is a known issue with Python's recognition of Windows ports above 10, but ampy does not take this into account, so we must format our port explicity for Python. Try formatting your COM port as you see it here (replacing <PORT> with your port number/location): \\.\<PORT>

If the file was uploaded, you should not see any error messages from ampy. If you do see some error messages, check out the [Troubleshooting section](https://learn.sparkfun.com/tutorials/micropython-programming-tutorial-getting-started-with-the-esp32-thing/troubleshooting) for some tips.

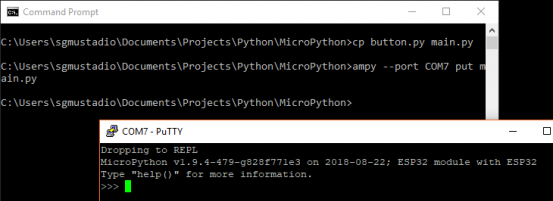


After uploading your code, it should automatically start running. However, because our code simply exits on a button push, there is nothing to see here.

Open a serial terminal and create a connection to your ESP32 Thing. You should see a blank terminal window, and even if you try typing some characters, nothing should appear.

Press the *0* button on your ESP32 Thing board.

As soon as you do, you should see a message appear in your serial terminal saying "Dropping to REPL" followed by the MicroPython version number and a command prompt (>>>). Note that we are using two separate programs here: the command terminal to upload the program and the serial terminal (I'm using PuTTY for my serial terminal) to communicate with REPL.



**Code to Note:**

Our program is relatively simple. We first import the machine, sys, and utime modules. [machine](http://docs.micropython.org/en/v1.7/esp8266/library/machine.html) allows us to control connected hardware, like pins. [sys](https://docs.micropython.org/en/latest/pyboard/library/sys.html) gives us access to underlying interpreter values and the ability to exit out of our forever loop in *main.py*. Finally, [utime](http://docs.micropython.org/en/latest/pyboard/library/utime.html?highlight=utime" \l "module-utime) allows us to get the local machine time (assuming we have a connected real-time clock) or tell our program to wait for a specified amount of time.

Much like in our REPL example, we create a machine.Pin object for pin 0, but we make it an input this time. We supply a third argument: machine.Pin.PULL\_UP. This says that we want to enable the ESP32's internal [pull-up resistor](https://learn.sparkfun.com/tutorials/pull-up-resistors) for pin 0. This allows the voltage on pin 0 to stay at 3.3 V until we push the onboard button, which subsequently pulls the voltage on the pin down to 0 V. We can use this technique to determine if the button has been pressed.

Everything indented under while True: runs forever. Each line is executed sequentially before returning to the top of the while loop.

Within the loop, we check to see if the button has been pressed:

COPY CODE**if** repl\_button.value() **==** 0:

print("Dropping to REPL")

sys.**exit**()

If it has, we print a message out to the serial terminal and then call sys.exit(), which raises a SystemExit exception. This will force the program to exit (and the MicroPython interpreter will perform any garbage collection or cleanup as necessary).

If no button push is detected, we tell the ESP32 to do nothing (sleep) for 1 ms:

COPY CODEutime.sleep\_ms(1)

This process is repeated again within the while loop. In future examples, we will continue using this "drop to REPL" button method to help us upload new code.

**Note:** If you have any problems running this code, refer to the [Troubleshooting section](https://learn.sparkfun.com/tutorials/micropython-programming-tutorial-getting-started-with-the-esp32-thing/troubleshooting) near the end of this tutorial.

Code Part 2: Blinky

In a text editor, enter the following code:

COPY CODE**import** machine

**import** sys

**import** utime

# Pin definitions

repl\_button **=** machine.Pin(0, machine.Pin.IN, machine.Pin.PULL\_UP)

led **=** machine.Pin(5, machine.Pin.OUT)

# Blink forever

**while** True:

# If button 0 is pressed, drop to REPL

**if** repl\_button.value() **==** 0:

print("Dropping to REPL")

sys.**exit**()

# Turn LED on and then off

led.value(1)

utime.sleep\_ms(500)

led.value(0)

utime.sleep\_ms(500)

Save the code with a name like *blink.py*

**Run It:**

Before uploading our code, we need to first stop our first program. **Close out of the serial terminal** (this is important to free up the serial port) and **press button 0** on your ESP32 Thing (this is also important!). Remember: pressing button 0 stops our program and gives us a REPL prompt. ampy will need this to upload a new program.

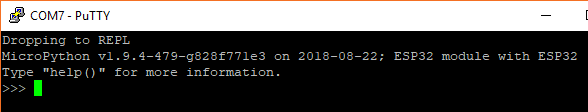
In a command terminal, copy *blink.py* and rename it as *main.py* (don't worry about overwriting your previous *main.py*, as we have the first example code saved as *button.py*). Don't forget to change <PORT> to your particular port number/location.

COPY CODEcp blink.py main.py

ampy --port **<**PORT**>** put main.py

The onboard blue LED should begin blinking on and off every second.

Open up a serial terminal and connect to your ESP32 Thing. Press and hold button 0 until the blinking LED stops. You should also see a REPL prompt appear.



**Code to Note:**

This time, we combined parts from our first REPL example and our button example. We added the following part in the while loop:

COPY CODEled.value(1)

utime.sleep\_ms(500)

led.value(0)

utime.sleep\_ms(500)

Much like we saw in the REPL example, we turn the LED on with led.value(1) and turn it off with led.value(0). We sleep 500 ms between each action so that we can see the LED blinking.

At the top of our while loop, we added our "drop to REPL" button press code to make debugging and uploading code on the ESP32 easier. Note that you need to hold down button 0 this time, as it can take over 1 second for one iteration of the while loop to complete. Execution needs to return to the beginning of the loop before checking for a button 0 press again.