HARDWARE EXAMPLES

The Particle Raspberry Pi project has been discontinued. You can still follow these instructions, however there will be no future updates and support is no longer available for this product.

Here you will find a bunch of examples to get you started using your Raspberry Pi with Particle!

To complete all the examples, you will need the following materials:

Materials

- **Hardware**
  - Your Raspberry Pi 2 or 3
  - USB to micro USB cable
  - Power source for USB cable (such as your computer, USB battery, or power brick)
  - (2) Resistors between 220 Ohms and 1000 Ohms
  - (1) LED, any color
  - (1) Photoresistor or phototransistor (explained below)
  - (Optional) A break out board like the Adafruit Pi T-Cobbler

Blink an LED

Intro

Blinking an LED is the "Hello World" example of the microcontroller universe. It's a nice way to warm up and start your journey into the land of embedded hardware.

Depending on the kit you have, you may have one or more of the items above. Left to right:

- IR (infrared) LED. It has a blue-ish tint. Don't use that one here, as you can't see the light with the naked eye!
- White LED. It has a rounded top. You can use this instead of the red LED in the examples below.
- Red LED.
- Photo transistor. The top of this is flat and is not an LED, it's a light sensor. You'll use that later.

There may also be smaller red or green LEDs. Those are also fine.

Setup

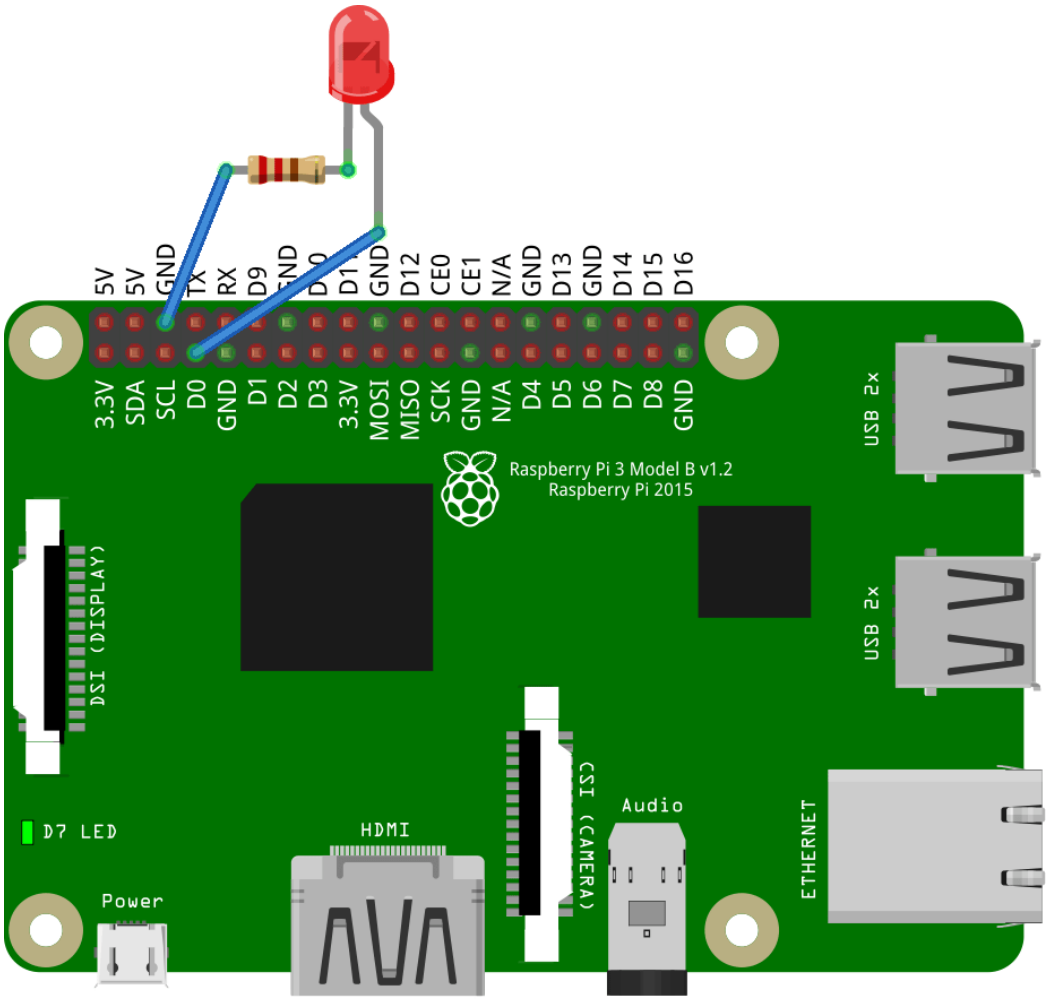
It's good practice to connect the red (+) bus bar on the top to 3V3 and the blue (-) bus bar on the bottom to ground.

- 3V3 is second from the left on the bottom (with the USB connector on the left). It is often connected using a red wire.
- Ground is the fourth from the left on the bottom. It is typically connected using a black wire.

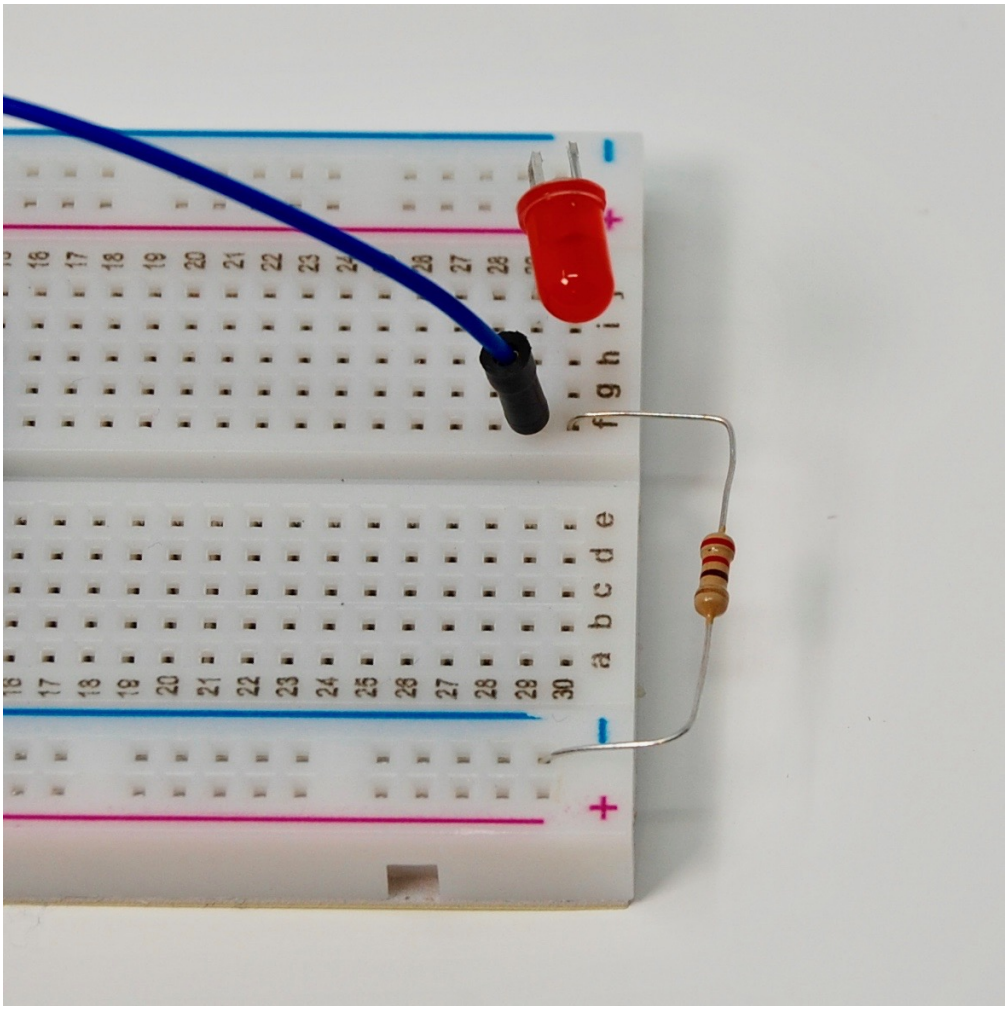
Position the LED in the breadboard. The long lead (anode) goes to + (left) and the short lead (cathode) goes to - (right). When using an LED, you must always add a current liming resistor. Normally you'd use a 220 ohm resistor (red-red-brown-gold) for 3.3 volt circuits.

In the picture, the long lead of the LED connects to pin D6 using the blue wire. The short lead of the LED connects to a 220 ohm resistor that connects it to ground. That completes the circuit.

Feel free to continue without connecting the external LED if that's too much trouble since the Raspberry Pi has a green LED on board that we'll use in the blink example.



Here's a close-up of the connections



- The long lead of the LED is on the left. The blue wire connects to this lead.
- The short lead of the LED is on the right. This connects to ground with a 220 ohm resistor.

Next, we're going to load code onto your device.

Code

Click this link to open the source code in the Particle Web IDE, or copy and paste the code below.

Go ahead and save this application, then flash it to your device. You should be able to see that LED blinking away!

```
// ------------
// Blink an LED
// ------------

/*-------------

We've heavily commented this code for you. If you're a pro, feel free to
ignore it.

Comments start with two slashes or are blocked off by a slash and a star.
You can read them, but your device can't.
It's like a secret message just for you.

Every program based on Wiring (programming language used by Arduino, and
Particle devices) has two essential parts:
setup - runs once at the beginning of your program
loop - runs continuously over and over

You'll see how we use these in a second.

This program will blink an led on and off every second.
It blinks the D7 LED on your Particle device. If you have an LED wired to D0,
it will blink that LED as well.

-------------*/


// First, we're going to make some variables.
// This is our "shorthand" that we'll use throughout the program:

int led1 = D6; // Instead of writing D0 over and over again, we'll write led1
// You'll need to wire an LED to this one to see it blink.

int led2 = D7; // Instead of writing D7 over and over again, we'll write led2
// This one is the little blue LED on your board. On the Photon it is next to
D7, and on the Core it is next to the USB jack.

// Having declared these variables, let's move on to the setup function.
// The setup function is a standard part of any microcontroller program.
// It runs only once when the device boots up or is reset.

void setup() {
```

```
    // We are going to tell our device that D0 and D7 (which we named led1 and
led2 respectively) are going to be output
    // (That means that we will be sending voltage to them, rather than
monitoring voltage that comes from them)

    // It's important you do this here, inside the setup() function rather
than outside it or in the loop function.

    pinMode(led1, OUTPUT);
    pinMode(led2, OUTPUT);

}

// Next we have the loop function, the other essential part of a
microcontroller program.
// This routine gets repeated over and over, as quickly as possible and as
many times as possible, after the setup function is called.
// Note: Code that blocks for too long (like more than 5 seconds), can make
weird things happen (like dropping the network connection).  The built-in
delay function shown below safely interleaves required background activity, so
arbitrarily long delays can safely be done if you need them.

void loop() {
    // To blink the LED, first we'll turn it on...
    digitalWrite(led1, HIGH);
    digitalWrite(led2, HIGH);

    // We'll leave it on for 1 second...
    delay(1000);

    // Then we'll turn it off...
    digitalWrite(led1, LOW);
    digitalWrite(led2, LOW);

    // Wait 1 second...
    delay(1000);

    // And repeat!
}
```

Control LEDs over the 'net

Intro

Now that we know how to blink an LED, how about we control it over the Internet? This is where the fun begins.

We've heavily commented the code below so that you can see what's going on. Basically, we are going to:

- Set up the pins as outputs that have LEDs connected to them
- Create and register a Particle function (this gets called automagically when you make an API request to it)
- Parse the incoming command and take appropriate actions

Setup

Set up the LED as in the previous example.

Code

Click this link to open the source code in the Particle Web IDE, or copy and paste the code below.

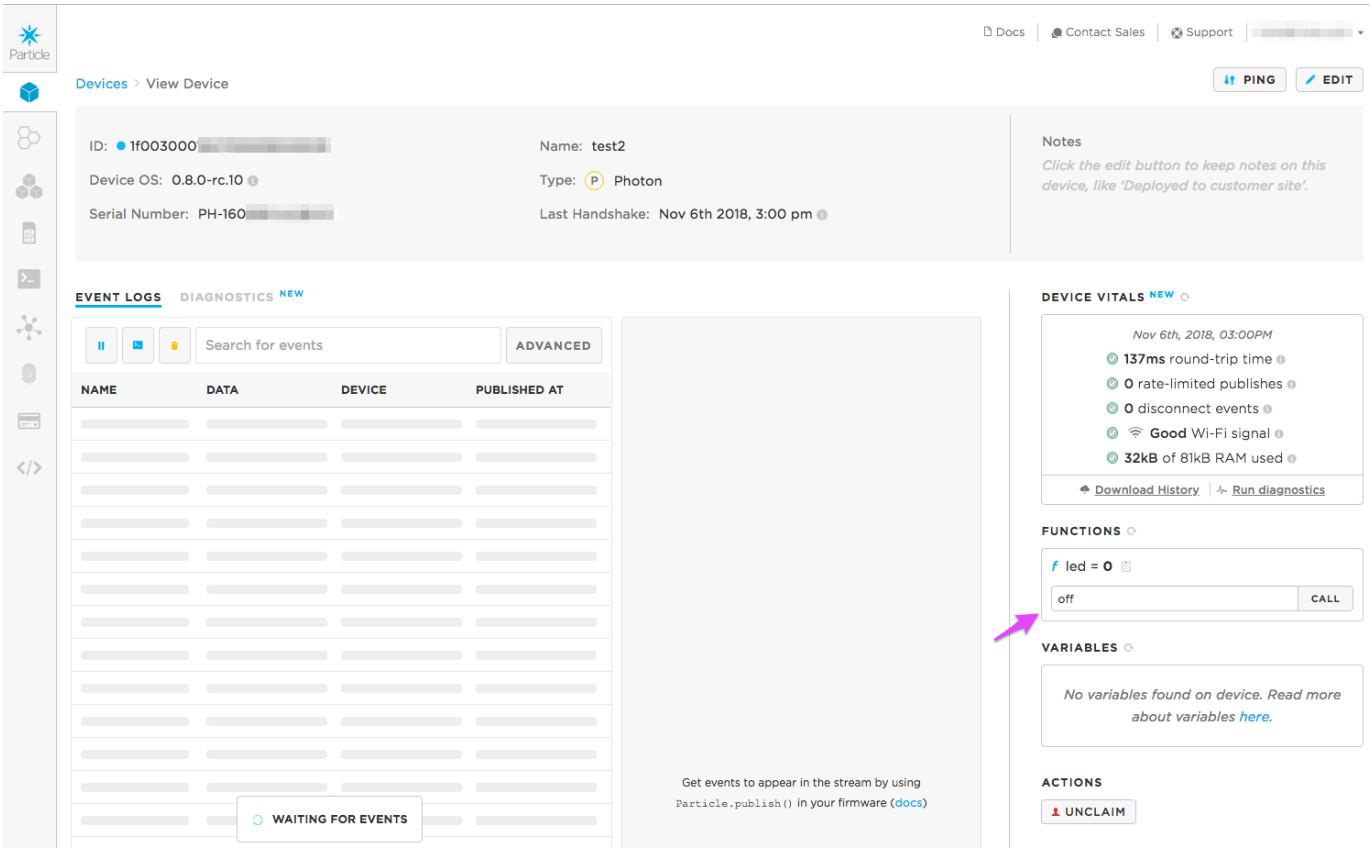
```
// ---------------------------------
// Controlling LEDs over the Internet
// ---------------------------------

/* First, let's create our "shorthand" for the pins
Same as in the Blink an LED example:
led1 is D6, led2 is D7 */

int led1 = D6;
int led2 = D7;
```

```
// Last time, we only needed to declare pins in the setup function.
// This time, we are also going to register our Particle function

void setup()
{

   // Here's the pin configuration, same as last time
   pinMode(led1, OUTPUT);
   pinMode(led2, OUTPUT);

   // We are also going to declare a Particle.function so that we can turn the
LED on and off from the cloud.
   Particle.function("led",ledToggle);
   // This is saying that when we ask the cloud for the function "led", it
will employ the function ledToggle() from this app.

   // For good measure, let's also make sure both LEDs are off when we start:
   digitalWrite(led1, LOW);
   digitalWrite(led2, LOW);

}



/* Last time, we wanted to continously blink the LED on and off
Since we're waiting for input through the cloud this time,
we don't actually need to put anything in the loop */

void loop()
{
   // Nothing to do here
}

// We're going to have a super cool function now that gets called when a
matching API request is sent
// This is the ledToggle function we registered to the "led" Particle.function
earlier.

int ledToggle(String command) {
    /* Particle.functions always take a string as an argument and return an
integer.
    Since we can pass a string, it means that we can give the program commands
on how the function should be used.
    In this case, telling the function "on" will turn the LED on and telling
it "off" will turn the LED off.
```

```
    Then, the function returns a value to us to let us know what happened.
    In this case, it will return 1 for the LEDs turning on, 0 for the LEDs
turning off,
    and -1 if we received a totally bogus command that didn't do anything to
the LEDs.
    */

    if (command=="on") {
        digitalWrite(led1,HIGH);
        digitalWrite(led2,HIGH);
        return 1;
    }
    else if (command=="off") {
        digitalWrite(led1,LOW);
        digitalWrite(led2,LOW);
        return 0;
    }
    else {
        return -1;
    }
}
```

Testing

The easiest way to test your new function is from the console. In the devices tab, click on the row for the device you just flashed your code to.



On the right-hand side of the screen is a box for functions. There should be one labeled "led". If you type "on" (without the quotes") and click the Call button the LED should turn on.

Use

When we register a function or variable, we're basically making a space for it on the internet, similar to the way there's a space for a website you'd navigate to with your browser. Thanks to the REST API, there's a specific address that identifies you and your device. You can send requests, like `GET` and `POST` requests, to this URL just like you would with any webpage in a browser.

Remember the last time you submitted a form online? You may not have known it, but the website probably sent a `POST` request with the info you put in the form over to another URL that would store your data. We can do the same thing to send information to your device, telling it to turn the LED on and off.

Use the following to view your page:

```
/* Paste the code between the dashes below into a .txt file and save it as an
.html file. Replace your-device-ID-goes-here with your actual device ID and
```

```
your-access-token-goes-here with your actual access token.

---------------------------
<!-- Replace your-device-ID-goes-here with your actual device ID
and replace your-access-token-goes-here with your actual access token-->
<!DOCTYPE>
<html>
  <body>
  <center>
  <br>
  <br>
  <br>
  <form action="https://api.particle.io/v1/devices/your-device-ID-goes-
here/led?access_token=your-access-token-goes-here" method="POST">
    Tell your device what to do!<br>
    <br>
    <input type="radio" name="arg" value="on">Turn the LED on.
    <br>
    <input type="radio" name="arg" value="off">Turn the LED off.
    <br>
    <br>
    <input type="submit" value="Do it!">
  </form>
  </center>
  </body>
</html>
---------------------------
*/
```

Edit the code in your text file so that "your-device-ID-goes-here" is your actual device ID, and "your-access-token-goes-here" is your actual access token. These things are accessible through your IDE at build.particle.io. Your device ID can be found in your Devices drawer (the crosshairs) when you click on the device you want to use, and your access token can be found in settings (the cogwheel).

Open that `.html` file in a browser. You'll see a very simple HTML form that allows you to select whether you'd like to turn the LED on or off.

When you click "Do it!" you are posting information to the URL `https://api.particle.io/v1/devices/your-device-ID-goes-here/led?access_token=your-access-token-goes-here`. The information you give is the `args`, or argument value, of `on` or

`off`. This hooks up to your `Particle.function` that we registered with the cloud in our firmware, which in turn sends info to your device to turn the LED on or off.

You'll get some info back after you submit the page that gives the status of your device and lets you know that it was indeed able to post to the URL. If you want to go back, just click "back" on your browser.

If you are using the command line, you can also turn the LED on and off by typing:

```
particle call device_name led on
```

and

```
particle call device_name led off
```

Remember to replace `device_name` with either your device ID or the nickname you made for your device when you set it up.

This does the same thing as our HTML page, but with a more slick interface.

The API request will look something like this:

```
POST /v1/devices/{DEVICE_ID}/led

# EXAMPLE REQUEST IN TERMINAL
# Core ID is 0123456789abcdef
# Your access token is 123412341234
curl https://api.particle.io/v1/devices/0123456789abcdef/led \
  -d access_token=123412341234 \
  -d arg=on
```

Note that the API endpoint is 'led', not 'ledToggle'. This is because the endpoint is defined by the first argument of Particle.function(), which is a string of characters, rather than the second argument, which is a function.

To better understand the concept of making API calls to your device over the cloud checkout the Cloud API reference.

Script Execution, Publish and the Console

Intro

One of the most common use cases for IoT is to bring valuable sensor data online to make it accessible from anywhere and to inform intelligent reactions. Is the room temperature in your den climbing too high? Time to lower the shades. Are your plants overwatered from summer storms? Best not turn on the sprinklers.

In this example, we'll show you how to collect sensor data from your Raspberry Pi (the internal temperature of your Pi's processor) and post that data to the Particle Console.

The program to access the temperature sensor is `vcgencmd measure_temp`. It returns a string like `temp=43.5'C`.

After flashing the code below to your Raspberry Pi, you can check out the results on your console at console.particle.io. When the CPU is very busy the temperature will go up.

You can also hook up publishes to IFTTT! More info here.

Setup

For this example, we'll use the internal CPU temperature sensor in the Raspberry Pi so just power it up.

Code

```
// ----------------------------------------
// Execute a script and publish the result
// ----------------------------------------

void setup() {
  // Nothing to set up here
}

void loop() {
  // Measure the CPU temperature by running the vcgencmd program
  Process proc = Process::run("vcgencmd measure_temp");
  // Wait for vcgencmd to finish
  proc.wait();
```

```
  // The output is temp=43.5'C so fast-forward until the the character = is
found
  proc.out().find("=");
  // Convert the string to a number
  float cpuTemp = proc.out().parseFloat();

  // Publish the event to the Particle cloud. It will be visible in the
Console.
  Particle.publish("cpu_temp", String(cpuTemp), PRIVATE);

  // Repeat after a 1 second pause
  delay(1000);
}
```

Tinker

Remember back when we were blinking lights and reading sensors with Tinker on the mobile app?

When you tap a pin on the mobile app, it sends a message up to the cloud. Your device is always listening to the cloud and waiting for instructions-- like "write D7 HIGH" or "read the voltage at A0".

Your device already knew how to communicate with the mobile app because of the firmware loaded onto your device as a default. We call this the Tinker firmware. It's just like the user firmware you've been loading onto your device in these examples. It's just that with the Tinker firmware, we've specified special `Particle.function`s that the mobile app knows and understands.

If your device is new, it already has the Tinker firmware on it. It's the default firmware stored on your device right from the factory. When you put your own user firmware on your device, you'll rewrite the Tinker firmware. (That means that your device will no longer understand commands from the Particle mobile app.) However, you can always get the Tinker firmware back on your device by running `particle-agen setup`.

The Tinker app is a great example of how to build a very powerful application with not all that much code. If you're a technical person, you can have a look at the latest release here.

I know what you're thinking: this is amazing, but I really want to use Tinker *while* my code is running so I can see what's happening! Now you can.

Combine your code with this framework, flash it to your device, and Tinker away. You can also access Tinker code by clicking on the last example in the online IDE's code menu.

```
/* Function prototypes -------------------------------------------------------
*/
int tinkerDigitalRead(String pin);
int tinkerDigitalWrite(String command);
int tinkerAnalogRead(String pin);
int tinkerAnalogWrite(String command);

SYSTEM_MODE(AUTOMATIC);

/* This function is called once at start up ----------------------------------
*/
void setup()
{
    //Setup the Tinker application here

    //Register all the Tinker functions
    Particle.function("digitalread", tinkerDigitalRead);
    Particle.function("digitalwrite", tinkerDigitalWrite);

    Particle.function("analogread", tinkerAnalogRead);
    Particle.function("analogwrite", tinkerAnalogWrite);
}

/* This function loops forever ---------------------------------------------*/
void loop()
{
    //This will run in a loop
}

/*******************************************************************************
**
 * Function Name  : tinkerDigitalRead
 * Description    : Reads the digital value of a given pin
 * Input          : Pin
 * Output         : None.
 * Return         : Value of the pin (0 or 1) in INT type
```

```
                    Returns a negative number on failure

*******************************************************************************
*/
int tinkerDigitalRead(String pin)
{
    //convert ASCII to integer
    int pinNumber = pin.charAt(1) - '0';
    //Sanity check to see if the pin numbers are within limits
    if (pinNumber < 0 || pinNumber > 7) return -1;

    if(pin.startsWith("D"))
    {
        pinMode(pinNumber, INPUT_PULLDOWN);
        return digitalRead(pinNumber);
    }
    else if (pin.startsWith("A"))
    {
        pinMode(pinNumber+10, INPUT_PULLDOWN);
        return digitalRead(pinNumber+10);
    }
#if Wiring_Cellular
    else if (pin.startsWith("B"))
    {
        if (pinNumber > 5) return -3;
        pinMode(pinNumber+24, INPUT_PULLDOWN);
        return digitalRead(pinNumber+24);
    }
    else if (pin.startsWith("C"))
    {
        if (pinNumber > 5) return -4;
        pinMode(pinNumber+30, INPUT_PULLDOWN);
        return digitalRead(pinNumber+30);
    }
#endif
    return -2;
}

/*******************************************************************************
**
 * Function Name  : tinkerDigitalWrite
 * Description    : Sets the specified pin HIGH or LOW
 * Input          : Pin and value
 * Output         : None.
```

```
 * Return          : 1 on success and a negative number on failure

*******************************************************************************
*/
int tinkerDigitalWrite(String command)
{
    bool value = 0;
    //convert ASCII to integer
    int pinNumber = command.charAt(1) - '0';
    //Sanity check to see if the pin numbers are within limits
    if (pinNumber < 0 || pinNumber > 7) return -1;

    if(command.substring(3,7) == "HIGH") value = 1;
    else if(command.substring(3,6) == "LOW") value = 0;
    else return -2;

    if(command.startsWith("D"))
    {
        pinMode(pinNumber, OUTPUT);
        digitalWrite(pinNumber, value);
        return 1;
    }
    else if(command.startsWith("A"))
    {
        pinMode(pinNumber+10, OUTPUT);
        digitalWrite(pinNumber+10, value);
        return 1;
    }
#if Wiring_Cellular
    else if(command.startsWith("B"))
    {
        if (pinNumber > 5) return -4;
        pinMode(pinNumber+24, OUTPUT);
        digitalWrite(pinNumber+24, value);
        return 1;
    }
    else if(command.startsWith("C"))
    {
        if (pinNumber > 5) return -5;
        pinMode(pinNumber+30, OUTPUT);
        digitalWrite(pinNumber+30, value);
        return 1;
    }
#endif
```

```
    else return -3;
}

/*******************************************************************************
**
 * Function Name  : tinkerAnalogRead
 * Description    : Reads the analog value of a pin
 * Input          : Pin
 * Output         : None.
 * Return         : Returns the analog value in INT type (0 to 4095)
                    Returns a negative number on failure

*******************************************************************************
*/
int tinkerAnalogRead(String pin)
{
    //convert ASCII to integer
    int pinNumber = pin.charAt(1) - '0';
    //Sanity check to see if the pin numbers are within limits
    if (pinNumber < 0 || pinNumber > 7) return -1;

    if(pin.startsWith("D"))
    {
        return -3;
    }
    else if (pin.startsWith("A"))
    {
        return analogRead(pinNumber+10);
    }
#if Wiring_Cellular
    else if (pin.startsWith("B"))
    {
        if (pinNumber < 2 || pinNumber > 5) return -3;
        return analogRead(pinNumber+24);
    }
#endif
    return -2;
}

/*******************************************************************************
**
 * Function Name  : tinkerAnalogWrite
 * Description    : Writes an analog value (PWM) to the specified pin
 * Input          : Pin and Value (0 to 255)
```

```
 * Output         : None.
 * Return         : 1 on success and a negative number on failure

*******************************************************************************
*/
int tinkerAnalogWrite(String command)
{
    String value = command.substring(3);

    if(command.substring(0,2) == "TX")
    {
        pinMode(TX, OUTPUT);
        analogWrite(TX, value.toInt());
        return 1;
    }
    else if(command.substring(0,2) == "RX")
    {
        pinMode(RX, OUTPUT);
        analogWrite(RX, value.toInt());
        return 1;
    }

    //convert ASCII to integer
    int pinNumber = command.charAt(1) - '0';
    //Sanity check to see if the pin numbers are within limits

    if (pinNumber < 0 || pinNumber > 7) return -1;

    if(command.startsWith("D"))
    {
        pinMode(pinNumber, OUTPUT);
        analogWrite(pinNumber, value.toInt());
        return 1;
    }
    else if(command.startsWith("A"))
    {
        pinMode(pinNumber+10, OUTPUT);
        analogWrite(pinNumber+10, value.toInt());
        return 1;
    }
    else if(command.substring(0,2) == "TX")
    {
        pinMode(TX, OUTPUT);
        analogWrite(TX, value.toInt());
```

```
        return 1;
    }
    else if(command.substring(0,2) == "RX")
    {
        pinMode(RX, OUTPUT);
        analogWrite(RX, value.toInt());
        return 1;
    }
#if Wiring_Cellular
    else if (command.startsWith("B"))
    {
        if (pinNumber > 3) return -3;
        pinMode(pinNumber+24, OUTPUT);
        analogWrite(pinNumber+24, value.toInt());
        return 1;
    }
    else if (command.startsWith("C"))
    {
        if (pinNumber < 4 || pinNumber > 5) return -4;
        pinMode(pinNumber+30, OUTPUT);
        analogWrite(pinNumber+30, value.toInt());
        return 1;
    }
#endif
    else return -2;
}
```