31. Bluetooth Examples

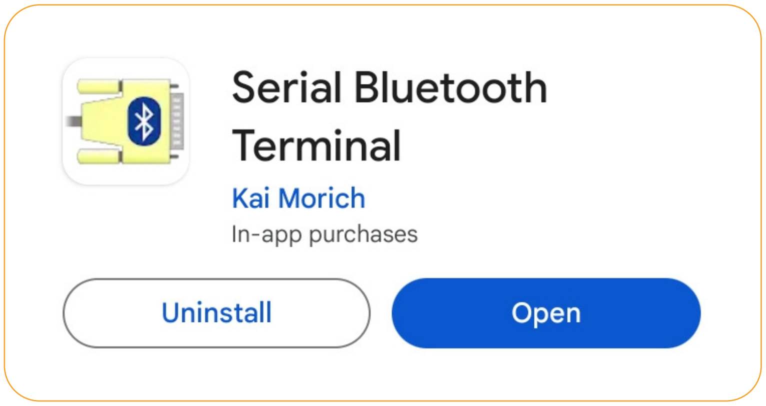
Introduction:

In the previous lecture, data was sent do and from the Pi-Pico using a cell phone with a bluetooth connection. In this lecture, we'll build upon this to allow you to use your cell phone to:

- Control the brightness of a NeoPixel flashlight from 0% to 100%,
- Control the individual colors of a NeoPixel flashlight from 0% to 100%,
- Control the on and off times for a strobe light, and
- Set the timing, start, and stop a starter tree

through this bluetooth connection.

All examples in this lecture assume you're using the Serial Bluetooth Terminal cell phone app by Kai Morich.



Cell Phone App: Serial Bluetooth Terminal app by Kai Morich

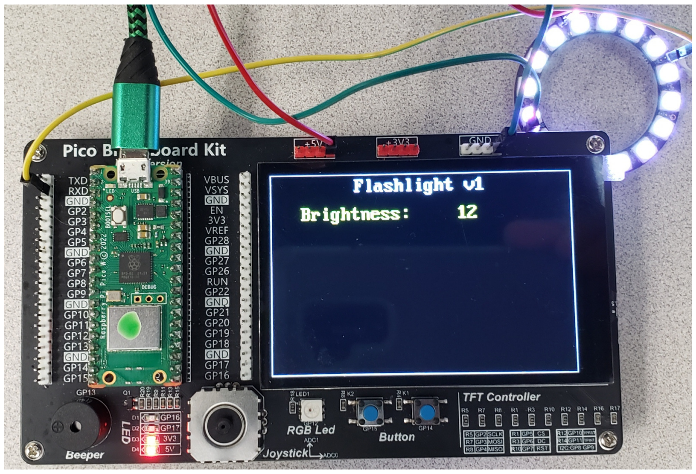
Once this app is installed on your cell phone and the drivers

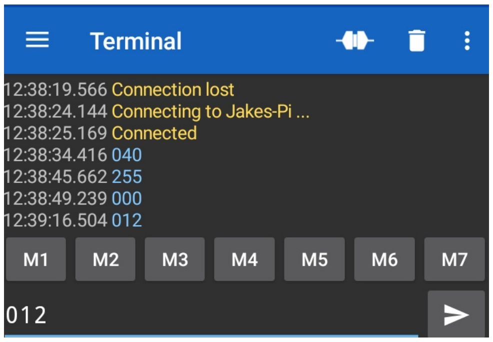
- *ble_advertising.py*
- *ble_simple_peripheral.py*

are downloaded to your Pico board, you can start sending and receiving data from your cell phone using bluetooth. Please refer to lecture #30 for details on how to make this bluetooth connection using this app.

NeoPixel Brightness Control

- Flashlight v1.py





```
12:38:19.566 Connection lost
12:38:24.144 Connecting to Jakes-Pi ...
12:38:25.169 Connected
12:38:34.416 040
12:38:45.662 255
12:38:49.239 000
12:39:16.504 012
012
```

Starting out, let's program the Pico to

- Control the brightness of a NeoPixel,
- Controlled by your cell phone
- Through a bluetooth interface.

Assume all LEDs display white light (red / green / blue are the same) with levels from 0 to 255.

The first step is to choose a method for driving the NeoPixels. Previous lectures gave several options for doing so. For convenience (and because it works for this particular NeoPixel string), let's use the neopixel library that comes with Thonny. The base code to drive a NeoPixel with 16 elements is as follows:

```
import machine, neopixel

N = 16
p = machine.Pin(1)
np = neopixel.NeoPixel(p, N, bpp=3, timing=1)

np.fill([0,1,2])
np.write()
```

base code for driving a 16-element NeoPixel connected to GP1

Next, choose the format for the data that's sent from your cell phone. For convenience, assume the data is

- Three digits of ASCII text
- Ranging from 000 to 255

Using code from the previous lecture, sending a message such as '012' from your cell phone results in the Pico receiving the binary string

```
data = b'012\r\n'
```

This can be converted to an integer using the command

```
        Level = int(data[0:3])
```

Once the number (000 to 255) is pulled from the binary string, the brightness of the NeoPixel can be set:

```
def on_rx(data):
    global Level, flag
    print("Data received: ", data)
    try:
        Level = int(data[0:3])
        Level = min(Level, 255)
        print('Brightness = ',Level)
        np.fill([Level,Level,Level])
        np.write()
        flag = 1
    except:
        print('invalid data entry')
```

BlueTooth receive routine: the brightness level is pulled from the binary string passed in *data*

The main routine can then wait for a message to come in from the bluetooth connection. Once it comes in,

- The message is passed to the on_rx() routine (which pulls out the brightness level), and
- The LCD display is updated to show the brightness of the NeoPixel flashlight

Note that

- A flag is used to tell the main loop when the data has received and the LCD display needs to be updated, and
- A global variable (Level) is used to pass data from the main routine and the bluetooth receive routine.

You could probably use other methods to pass the brightness level as well if you wanted.

```
flag = 1
Level = 0
np.fill([Level,Level,Level])
np.write()

while(1):
    if sp.is_connected():
        sp.on_write(on_rx)
    if(flag):
        LCD.Number2(Level, 3, 0, 300, 50, Yellow, Black)
        flag = 0
```

Main routine: Data is passed to the on_rx() routine when a message is received from the bluetooth connection.
The LCD display is also updated when this happens.

The shell window displays debug information as well. This includes

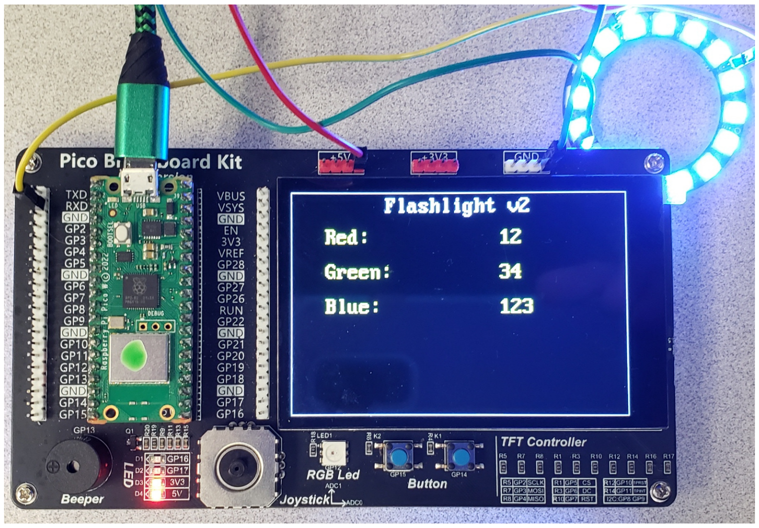
- The name of the bluetooth connection (64)
- The raw data received, and
- The brightness level pulled from the bluetooth data.

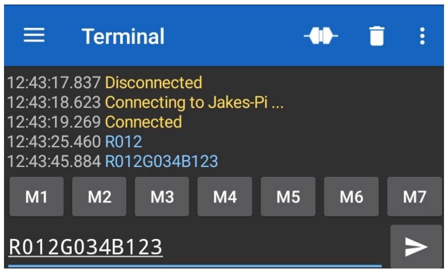
```
Starting advertising
New connection 64
Data received:  b'040\r\n'
Brightness =  40
Data received:  b'255\r\n'
Brightness =  255
Data received:  b'000\r\n'
Brightness =  0
```

Shell Window showing the bluetooth connection has been made as well as the raw data received

NeoPixel Flashlight (take 2)

- Flashlight v2.py





```
12:43:17.837 Disconnected
12:43:18.623 Connecting to Jakes-Pi ...
12:43:19.269 Connected
12:43:25.460 R012
12:43:45.884 R012G034B123
```

In the previous code, only white light could be displayed (red / green / blue values were set to the same level). In this example, the hue of the NeoPixel flashlight can be set by setting the brightness level of red, green, and blue independently.

One of the first things that needs to be defined is how the data from your cell phone is to be formatted. This is somewhat arbitrary and up to the programmer.

Assume here that the RGB levels are send as a 12-character string:

```
  RxxxGyyyBzzz
```

where xxx, yyy, and zzz are the brightness levels of red, green, and blue respectively. For the convenience of the programmer, assume each of these numbers are three digits (for example, to turn off all LEDs, R0G0B0 isn't valid - you have to input this as R000G000B000).

This requirement simplifies the parsing of the data:

- The red level is always the 3-digit number located at [1,3]
- The blue level is always the 3-digit number located at [5,7], and
- The green level is always the 3-digit number at [9,11]

The routine to receive and parse the data is then as follows. Global variables are again used to pass data back to the main routine (other options exist)

```
def on_rx(data):
    global r, g, b, flag
    print("Data received: ", data)
    try:
        r = int(data[1:4])
        g = int(data[5:8])
        b = int(data[9:12])
        print('red = ',r,' green = ',g,'  blue = ',b)
        np.fill([r,g,b])
        np.write()
        flag = 1
    except:
        print('format for data is RxxxGxxxBxxx')
```

Data reception routine

The main routine then waits until a message is received from the bluetooth connection. Once received,

- The raw data is passed to the on_rx routine which extracts the r/g/b levels
- The LCD display is then updated showing the hue of the flashlight.

```
flag = 1

while(1):
    if sp.is_connected():
        sp.on_write(on_rx)
    if(flag):
        LCD.Text2(str(r) + '   ', 300, 50, Yellow, Black)
        LCD.Text2(str(g) + '   ', 300, 100, Yellow, Black)
        LCD.Text2(str(b) + '   ', 300, 150, Yellow, Black)
        flag = 0
```

Main route: pass data to the on_rx() routine when a bluetooth message is received

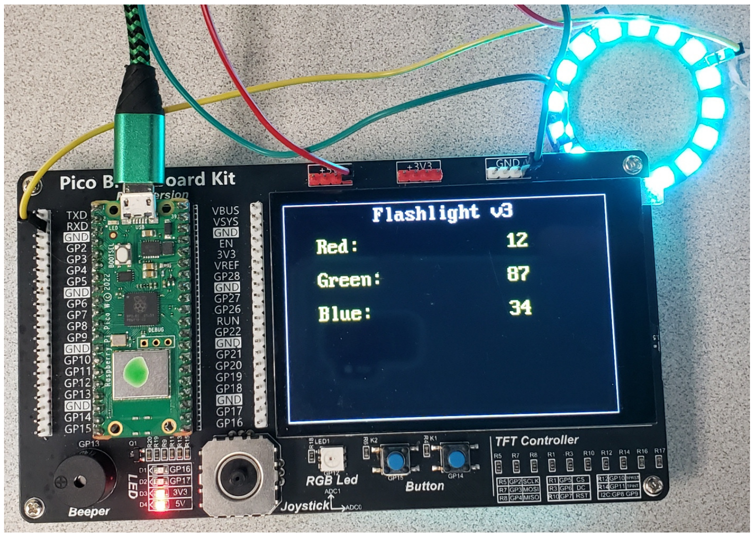
The shell window is just used for debugging. This shows

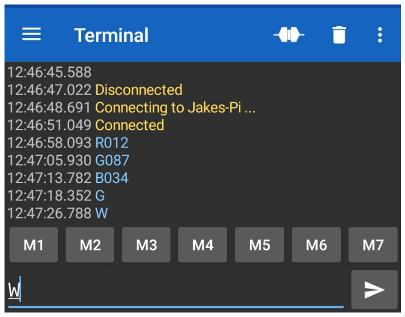
- The raw message received from your cell phone, and
- The resulting red / green / blue levels which are pulled from this message.

```
-------------------------------
Data received:  b'R012G034B078\r\n'
red =  12  green =  34   blue =  78
Data received:  b'R000G000B000\r\n'
red =  0  green =  0   blue =  0
```

Shell window displays debug information including the raw data received and the resulting rgb levels

NeoPixel Flashlight (take 3)





```
12:46:45.588
12:46:47.022 Disconnected
12:46:48.691 Connecting to Jakes-Pi ...
12:46:51.049 Connected
12:46:58.093 R012
12:47:05.930 G087
12:47:13.782 B034
12:47:18.352 G
12:47:26.788 W
W
```

In the previous example, a single string such as

```
R012G102B214
```

was required each time you update the NeoPixel. Another option is to define a set of instructions you can send. For example, to set the brightness, three commands could be used to set the brightens of each color:

```
Rxxx
Gyyy
Bzzz
```

The Pico looks at the first character to determine which color is being addressed. Once determined, the next three characters are read to determine the corresponding brightness.

Two more commands are added

```
C        sets r=g=b=0 and turns off the flashlight
W        writes the current value of r/g/b to the NeoPixel
```

What instructions you choose and how you define them is really up to the programmer.

Using these five instructions, the *on_rx()* routine can pull out the appropriate data. Note again that global variables are used to pass data to the main routine.

```
def on_rx(data):
    global r,g,b, flag
    print("Data received: ", data)
    try:
        cmd = chr(data[0])
        if(cmd == 'R'):
            r = int(data[1:4])
        if(cmd == 'G'):
            g = int(data[1:4])
        if(cmd == 'B'):
            b = int(data[1:4])
            print('b = ', b)
        if(cmd == 'C'):
            r = g = b = 0
        if(cmd == 'W'):
            np.fill([r,g,b])
            np.write()
        flag = 1
    except:
        print ('data format is Rxxx / Gxxx / Bxxx / C / W ')
```

on_rx() routine to parse the data sent to the Pico

The main routine then

- Passes the text string to the on_rx() routine when received, and
- Displays the brightness on the LCD display

```
flag = 1
r=g=b=0
np.fill([r,g,b])
np.write()

while(1):
    if sp.is_connected():
        sp.on_write(on_rx)
    if(flag):
        LCD.Number2(r, 3, 0, 300, 50, Yellow, Black)
        LCD.Number2(g, 3, 0, 300, 100, Yellow, Black)
        LCD.Number2(b, 3, 0, 300, 150, Yellow, Black)
        flag = 0
```

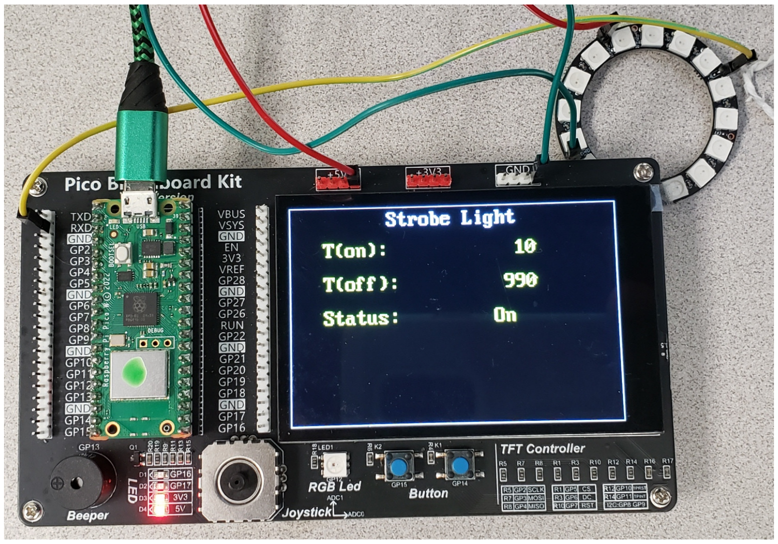
Main routine: pass data when received and display the resulting brightness.

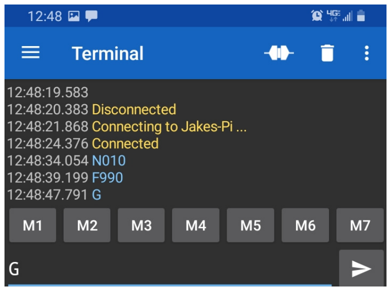
The shell window displays debug information, such as the raw data received

```
-------------------------

New connection 64
Data received:  b'\r\n'
Data received:  b'R012\r\n'
Data received:  b'G087\r\n'
Data received:  b'B034\r\n'
Data received:  b'W\r\n'
Data received:  b'C\r\n'
```

NeoPixel Strobe Light

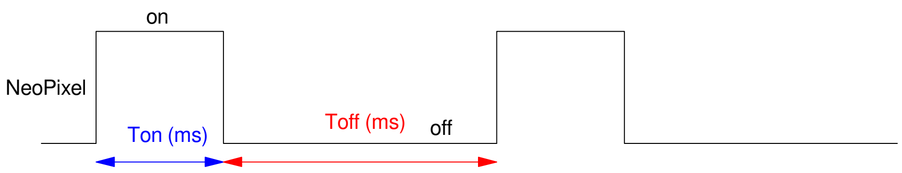




```
12:48:19.583
12:48:20.383 Disconnected
12:48:21.868 Connecting to Jakes-Pi ...
12:48:24.376 Connected
12:48:34.054 N010
12:48:39.199 F990
12:48:47.791 G
```

G

Yet another example of using your cell phone to control your Pico's operation is to turn your Pico into a strobe light. Here, the goal is to turn on and off a strobe light using your cell phone. In addition, the on-time and off-time of the strobe light should be controllable via your cell phone.



The on and off times can be controlled via your cell phone

Assume for this problem that

- All 16 NeoPixels are either off (00/00/00) or on (255/255/255)
- The on time can be adjusted from 1ms to 100ms
- The off time can be adjusted from 1ms to 100ms

Since timing is somewhat important for this problem, use timer interrupts for the on-time and off-time

- Each interrupt sets up the next interrupt T milliseconds in the future
- Interrupts are set up as a one-shot interrupt (since the on and off times are specified). If instead the period was specified, a periodic interrupt could be used to turn on the lights.
- The strobe light is turned off by skipping one of the one-shot initializations.

Another way to stop the strobe light would be to execute a *deinit()* command.

```
tim = Timer()

def Light_On(timer):
  global Ton, Status
  tim.init(period = Ton, mode=Timer.ONE_SHOT, callback=Light_Off)
  np.fill([255,255,255])
  np.write()

def Light_Off(timer):
  global Toff, Status
  if(Status):
    tim.init(period = Toff, mode=Timer.ONE_SHOT, callback=Light_On)
  np.fill([0,0,0])
  np.write()
```

Timer interrupts set the on and off times

Once the timing is set, the format of the data from your cell phone needs to be defined. Assume the data is sent as ASCII strings as:

- Nxxx sets the on time is milliseconds (000 to 999)
- Fxxx sets the off time in milliseconds (000 to 999)
- G turns on the strobe light (go), and
- S stops the strobe light (stop)

The *on_rx()* routine to parse the command accordingly is as follows:

```
def on_rx(data):
  global Ton, Toff, Status, flag
  print("Data received: ", data)
  try:
    cmd = chr(data[0])
    if(cmd == 'N'):
      Ton = int(data[1:4])
    if(cmd == 'F'):
      Toff = int(data[1:4])
    if(cmd == 'G'):
      Status = 1
      tim.init(period = Ton, mode=Timer.ONE_SHOT, callback=Light_On)
    if(cmd == 'S'):
      Status = 0
    flag = 1
  except:
    print('invalid data entry')
    print('Nxxx     set the on time in ms')
    print('Fxxx     set the off time in ms')
    print('G        start the strobe light')
    print('S        stop the strobe light')
```

on_rx() routine to parse the data coming in

The main routine then

- Passes data to on_rx() when a bluetooth message is received, and
- Displays the status of the strobe light.

Note that global variables are used to pass data from the on_rx() routine and the main routine. Other methods to pass data could also be used.

```
np.fill([0,0,0])
np.write()
flag = 1

while(1):
    if sp.is_connected():
        sp.on_write(on_rx)
    if(flag):
        LCD.Number2(Ton, 3, 0, 300, 50, Yellow, Black)
        LCD.Number2(Toff, 3, 0, 300, 100, Yellow, Black)
        if(Status == 1):
            LCD.Text2('On ', 300, 150, Yellow, Black)
        else:
            LCD.Text2('Off', 300, 150, Yellow, Black)
        flag = 0
```

on_rx() routine to parse the incoming data

Finally, debug information is displayed in the shell window. This shows the raw messages received from your cell phone.

```
-------------------

Data received:  b'N010\r\n'
Data received:  b'F490\r\n'
Data received:  b'G\r\n'
Data received:  b'S\r\n'
```

Shell window shows the raw messages received on the bluetooth link

Summary

Using the libraries

- *ble_advertising.py*
- *ble_simple_peripheral.py*

you are able to send data from your cell phone to your Pi-Pico. With a little coding, different commands can be sent to the Pico, controlling its operation, such as the brightness, color, or flashing rate of a NeoPixel. Other functions and commands are possible and only limited by the imagination of the programmer.

References

https://electrocredible.com/raspberry-pi-pico-w-bluetooth-ble-micropython