

Lesson 10 Light up the WS2812

10.1 Overview

This lesson focuses on teaching how to control WS2812 RGB LEDs using a Raspberry Pi with the Adeept Robot HAT V3.2. It covers the necessary components, wiring, demonstration of running programs for different lighting effects (breathing light and flowing lights), and an introduction to the relevant code. By the end of this lesson, learners will be able to understand the basic principles of WS2812 LED control and operate the lighting effects on their own.

10.2 Required Components

| Components | Quantity | Picture |
|-----------------------|----------|---|
| Raspberry Pi | 1 |  |
| Adeept Robot HAT V3.2 | 1 |  |
| 3 pin Cable | 1 |  |
| WS2812 RGB LED | 1 |  |

10.3 Principle Introduction

The WS2812 is a highly versatile addressable RGB LED, featuring an integrated control circuit and RGB chips. This integration allows it to receive serial data signals, which are then used to precisely control the color and brightness of each individual LED within a strip or array.

Each WS2812 LED interprets the incoming data based on a specific format. The data is structured to define the intensity of red, green, and blue light components for each LED. Since the human eye perceives color through the combination of these three primary colors (RGB), by adjusting the values of red, green, and blue within a 0 - 255 range (where 0 represents no light emission and 255 represents maximum intensity), an extensive palette of over 16 million color combinations can be achieved. For instance, a value of [255, 0, 0] would result in a pure red color, [0, 255, 0] in pure green, and [0, 0, 255] in pure blue. Mixing different values of these components creates various hues, saturations, and brightness levels.

On the Adeept Robot HAT V3.2, communication with the WS2812 LEDs is established through the SPI (Serial Peripheral Interface) protocol. When the Raspberry Pi generates data to control the WS2812 LEDs, it first transmits this data to the Adeept Robot HAT V3.2. The HAT then acts as a bridge, forwarding the received data to the WS2812 LEDs in the format they can understand. This seamless data transfer enables the programming of each LED in the WS2812 setup to display a specific color according to the data sequence sent, facilitating the creation of dynamic and colorful lighting effects.

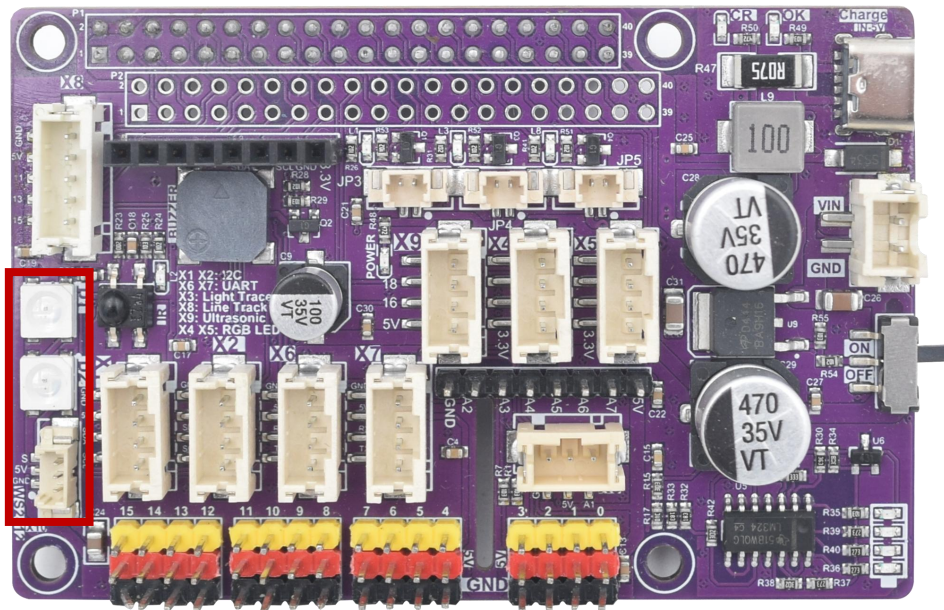
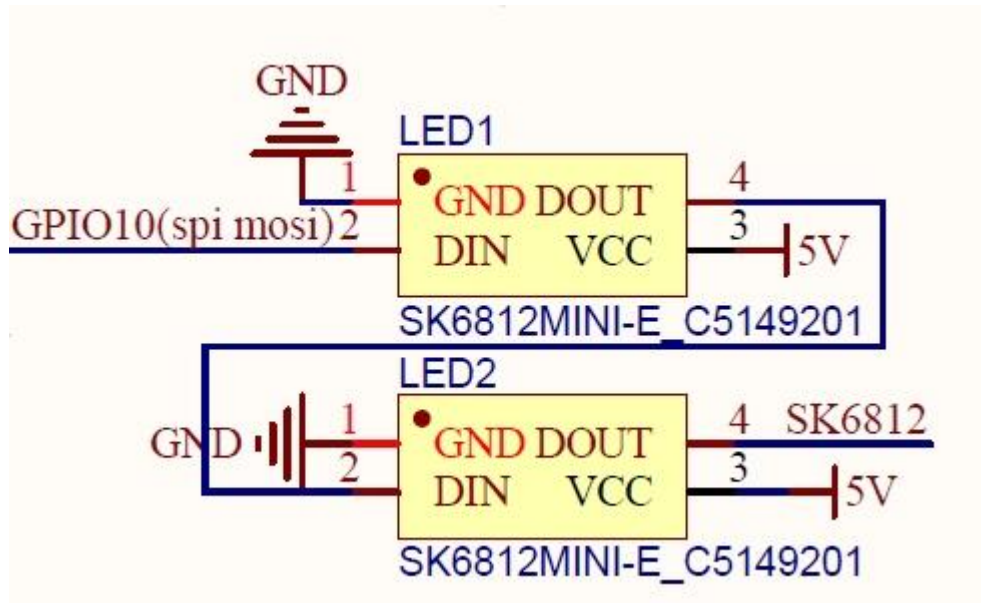
| PINS of Raspberry Pi | Sensor |
|----------------------|--------|
| GPIO10 | WS2812 |

10.4 Wiring Diagram

When using the WS2812 LED, the IN port of the LED must be connected to the WS2812 port on the Adeept Robot HAT V3.2 driver board.

Adeeps Robot HAT V3.2 has 2 built - in WS2812 LEDs, which are located near the GPIO10 pin at the front. When using the WS2812 Port interface to connect one or more external WS2812 LED modules, the first and second (indexed as 0 and 1 in the code) are used to control the two on -

board WS2812 LEDs. Starting from the third LED (index 2 in the code), they are used to control the externally extended WS2812 LEDs (indexed as 2, 3, 4, ...).



10.5 Demonstration

1. **Remotely log:** Remotely log in to the Raspberry Pi terminal.
2. **Navigate to the Program Folder:** Enter the following command in the terminal and press Enter to access the folder where the program is located:

```
cd Adeept_RaspClaws-V3/Examples/04_WS2812/
```

```
pi@raspberrypi:~ $ cd Adeept_RaspClaws-V3/Examples/04_WS2812/  
pi@raspberrypi:~/Adeept_RaspClaws-V3/Examples/04_WS2812 $
```

3. **View Directory Contents:** Type "ls" in the terminal and press Enter. This will display all the files in the current directory, ensuring that the "**BreathingLight.py**" and "**FlowingLights.py**" file is present:

```
ls
```

```
pi@raspberrypi:~/Adeept_RaspClaws-V3/Examples/04_WS2812 $ ls  
BreathingLight.py  FlowingLights.py
```

4. **Run the Program:** Enter the command below and press Enter to start the **BreathingLight.py** program:

```
sudo python3 BreathingLight.py
```

```
pi@raspberrypi:~/Adeept_RaspClaws-V3/Examples/04_WS2812 $ sudo python3 Breathing  
Light.py  
spidev version is 3.5  
spidev device as show:  
/dev/spidev0.0 /dev/spidev0.1 /dev/spidev10.0
```

5. **Observation and Termination:** The program prints the version of the spidev library and available SPI devices, creates an instance of the Adeept_SPI_LedPixel class and starts the thread, makes the WS2812 lights enter the breathing mode, and then goes into a dormant state waiting. To stop the running program, simply press the "**Ctrl + C**" shortcut on the keyboard. Call relevant methods to pause the thread, close the SPI connection, and turn off the WS2812 lights to release resources.

6. **Run the Program:** Enter the command below and press Enter to start the **FlowingLights.py** program:

```
sudo python3 FlowingLights.py
```

```
pi@raspberrypi:~/Adeept_RaspClaws-V3/Examples/04_WS2812 $ sudo python3 FlowingLi
ghts.py
```

7. Observation and Termination: The program prints the version of the spidev library and available SPI devices, creates an instance of the Adeept_SPI_LedPixel class and starts the thread, makes the WS2812 lights enter the chasing lights mode, and then goes into a dormant state waiting. To stop the running program, simply press the "**Ctrl + C**" shortcut on the keyboard. Call relevant methods to pause the thread, close the SPI connection, and turn off the WS2812 lights to release resources.

10.6 Code

Complete code refer to [BreathingLight.py](#)

```
001  #!/usr/bin/env/python
002  # File name   : BreathingLight.py
003  # Website    : www.Adeept.com
004  # Author     : Adeept
005  # Date      : 2025/04/8
006  import spidev
007  import sys
008  import subprocess
009  import threading
010  import numpy
011  from numpy import sin, cos, pi
012  import time
013  class Adeept_SPI_LedPixel(threading.Thread):
014      def __init__(self, count = 8, bright = 255, sequence='GRB', bus = 0, device = 0, *args, **kwargs):
015          self.set_led_type(sequence)
016          self.set_led_count(count)
017          self.set_led_brightness(bright)
018          self.led_begin(bus, device)
019          self.lightMode = 'none'
020          self.colorBreathR = 0
021          self.colorBreathG = 0
022          self.colorBreathB = 0
023          self.breathSteps = 10
024          self.set_all_led_color(0,0,0)
025          super(Adeept_SPI_LedPixel, self).__init__(*args, **kwargs)
026          self.__flag = threading.Event()
027          self.__flag.clear()
028      def led_begin(self, bus = 0, device = 0):
029          self.bus = bus
030          self.device = device
031          try:
032              self.spi = spidev.SpiDev()
033              self.spi.open(self.bus, self.device)
```

```

034         self.spi.mode = 0
035         self.led_init_state = 1
036     except OSError:
037         print("Please check the configuration in /boot/firmware/config.txt.")
038         if self.bus == 0:
039             print("You can turn on the 'SPI' in 'Interface Options' by using 'sudo raspi-config'.")
040             print("Or make sure that 'dtoverlay=spi=on' is not commented, then reboot the Raspberry
041 Pi. Otherwise spi0 will not be available.")
042         else:
043             print("Please add 'dtoverlay=spi{}-2cs' at the bottom of the /boot/firmware/config.txt,
044 then reboot the Raspberry Pi. otherwise spi{} will not be available.".format(self.bus, self.bus))
045         self.led_init_state = 0
046
047     def check_spi_state(self):
048         return self.led_init_state
049
050     def spi_gpio_info(self):
051         if self.bus == 0:
052             print("SPI0-MOSI: GPIO10(WS2812-PIN) SPI0-MISO: GPIO9 SPI0-SCLK: GPIO11 SPI0-CE0:
053 GPIO8 SPI0-CE1: GPIO7")
054             elif self.bus == 1:
055                 print("SPI1-MOSI: GPIO20(WS2812-PIN) SPI1-MISO: GPIO19 SPI1-SCLK: GPIO21 SPI1-CE0:
056 GPIO18 SPI1-CE1: GPIO17 SPI0-CE1: GPIO16")
057             elif self.bus == 2:
058                 print("SPI2-MOSI: GPIO41(WS2812-PIN) SPI2-MISO: GPIO40 SPI2-SCLK: GPIO42 SPI2-CE0:
059 GPIO43 SPI2-CE1: GPIO44 SPI2-CE1: GPIO45")
060             elif self.bus == 3:
061                 print("SPI3-MOSI: GPIO2(WS2812-PIN) SPI3-MISO: GPIO1 SPI3-SCLK: GPIO3 SPI3-CE0:
062 GPIO0 SPI3-CE1: GPIO24")
063             elif self.bus == 4:
064                 print("SPI4-MOSI: GPIO6(WS2812-PIN) SPI4-MISO: GPIO5 SPI4-SCLK: GPIO7 SPI4-CE0:
065 GPIO4 SPI4-CE1: GPIO25")
066             elif self.bus == 5:
067                 print("SPI5-MOSI: GPIO14(WS2812-PIN) SPI5-MISO: GPIO13 SPI5-SCLK: GPIO15 SPI5-CE0:
068 GPIO12 SPI5-CE1: GPIO26")
069             elif self.bus == 6:
070                 print("SPI6-MOSI: GPIO20(WS2812-PIN) SPI6-MISO: GPIO19 SPI6-SCLK: GPIO21 SPI6-CE0:
071 GPIO18 SPI6-CE1: GPIO27")
072
073     def led_close(self):
074         self.set_all_led_rgb([0,0,0])
075         self.spi.close()
076
077     def set_led_count(self, count):
078         self.led_count = count
079         self.led_color = [0,0,0] * self.led_count
080         self.led_original_color = [0,0,0] * self.led_count
081
082     def set_led_type(self, rgb_type):
083         try:
084             led_type = ['RGB', 'RBG', 'GRB', 'GBR', 'BRG', 'BGR']
085             led_type_offset = [0x06, 0x09, 0x12, 0x21, 0x18, 0x24]
086             index = led_type.index(rgb_type)
087             self.led_red_offset = (led_type_offset[index]>>4) & 0x03
088             self.led_green_offset = (led_type_offset[index]>>2) & 0x03
089             self.led_blue_offset = (led_type_offset[index]>>0) & 0x03

```

```
090         return index
091     except ValueError:
092         self.led_red_offset = 1
093         self.led_green_offset = 0
094         self.led_blue_offset = 2
095         return -1
096
097     def set_led_brightness(self, brightness):
098         self.led_brightness = brightness
099         for i in range(self.led_count):
100             self.set_led_rgb_data(i, self.led_original_color)
101
102     def set_ledpixel(self, index, r, g, b):
103         p = [0,0,0]
104         p[self.led_red_offset] = round(r * self.led_brightness / 255)
105         p[self.led_green_offset] = round(g * self.led_brightness / 255)
106         p[self.led_blue_offset] = round(b * self.led_brightness / 255)
107         self.led_original_color[index*3+self.led_red_offset] = r
108         self.led_original_color[index*3+self.led_green_offset] = g
109         self.led_original_color[index*3+self.led_blue_offset] = b
110         for i in range(3):
111             self.led_color[index*3+i] = p[i]
112
113     def set_led_color_data(self, index, r, g, b):
114         self.set_ledpixel(index, r, g, b)
115
116     def set_led_rgb_data(self, index, color):
117         self.set_ledpixel(index, color[0], color[1], color[2])
118
119     def set_led_color(self, index, r, g, b):
120         self.set_ledpixel(index, r, g, b)
121         self.show()
122
123     def set_led_rgb(self, index, color):
124         self.set_led_rgb_data(index, color)
125         self.show()
126
127     def set_all_led_color_data(self, r, g, b):
128         for i in range(self.led_count):
129             self.set_led_color_data(i, r, g, b)
130
131     def set_all_led_rgb_data(self, color):
132         for i in range(self.led_count):
133             self.set_led_rgb_data(i, color)
134
135     def set_all_led_color(self, r, g, b):
136         for i in range(self.led_count):
137             self.set_led_color_data(i, r, g, b)
138         self.show()
139
140     def set_all_led_rgb(self, color):
141         for i in range(self.led_count):
142             self.set_led_rgb_data(i, color)
143         self.show()
144
145     def write_ws2812_numpu8(self):
```



```

146         d = numpy.array(self.led_color).ravel()           #Converts data into a one-dimensional array
147         tx = numpy.zeros(len(d)*8, dtype=numpy.uint8)      #Each RGB color has 8 bits, each represented by
148         a uint8 type data
149         for ibit in range(8):                               #Convert each bit of data to the data that the
150         spi will send
151             tx[7-ibit::8]=((d>>ibit)&1)*0x78 + 0x80        #T0H=1,T0L=7, T1H=5,T1L=3   #0b111111000 mean
152         T1(0.78125us), 0b10000000 mean T0(0.15625us)
153             if self.led_init_state != 0:
154                 if self.bus == 0:
155                     self.spi.xfer(tx.tolist(), int(8/1.25e-6))    #Send color data at a frequency of
156         6.4Mhz
157                 else:
158                     self.spi.xfer(tx.tolist(), int(8/1.0e-6))    #Send color data at a frequency of
159         8Mhz
160
161     def write_ws2812_numpy4(self):
162         d=numpy.array(self.led_color).ravel()
163         tx=numpy.zeros(len(d)*4, dtype=numpy.uint8)
164         for ibit in range(4):
165             tx[3-ibit::4]=((d>>(2*ibit+1))&1)*0x60 + ((d>>(2*ibit+0))&1)*0x06 + 0x88
166         if self.led_init_state != 0:
167             if self.bus == 0:
168                 self.spi.xfer(tx.tolist(), int(4/1.25e-6))
169             else:
170                 self.spi.xfer(tx.tolist(), int(4/1.0e-6))
171
172     def show(self, mode = 1):
173         if mode == 1:
174             write_ws2812 = self.write_ws2812_numpy8
175         else:
176             write_ws2812 = self.write_ws2812_numpy4
177         write_ws2812()
178
179     def wheel(self, pos):
180         if pos < 85:
181             return [(255 - pos * 3), (pos * 3), 0]
182         elif pos < 170:
183             pos = pos - 85
184             return [0, (255 - pos * 3), (pos * 3)]
185         else:
186             pos = pos - 170
187             return [(pos * 3), 0, (255 - pos * 3)]
188
189     def hsv2rgb(self, h, s, v):
190         h = h % 360
191         rgb_max = round(v * 2.55)
192         rgb_min = round(rgb_max * (100 - s) / 100)
193         i = round(h / 60)
194         diff = round(h % 60)
195         rgb_adj = round((rgb_max - rgb_min) * diff / 60)
196         if i == 0:
197             r = rgb_max
198             g = rgb_min + rgb_adj
199             b = rgb_min
200         elif i == 1:
201             r = rgb_max - rgb_adj

```



```
202         g = rgb_max
203         b = rgb_min
204     elif i == 2:
205         r = rgb_min
206         g = rgb_max
207         b = rgb_min + rgb_adj
208     elif i == 3:
209         r = rgb_min
210         g = rgb_max - rgb_adj
211         b = rgb_max
212     elif i == 4:
213         r = rgb_min + rgb_adj
214         g = rgb_min
215         b = rgb_max
216     else:
217         r = rgb_max
218         g = rgb_min
219         b = rgb_max - rgb_adj
220     return [r, g, b]
221
222 def police(self):
223     self.lightMode = 'police'
224     self.resume()
225
226 def breath(self, R_input, G_input, B_input):
227     self.lightMode = 'breath'
228     self.colorBreathR = R_input
229     self.colorBreathG = G_input
230     self.colorBreathB = B_input
231     self.resume()
232
233 def resume(self):
234     self.__flag.set()
235
236 def pause(self):
237     self.lightMode = 'none'
238     self.set_all_led_color_data(0, 0, 0)
239     self.__flag.clear()
240
241
242 def breathProcessing(self):
243     while self.lightMode == 'breath':
244         for i in range(0, self.breathSteps):
245             if self.lightMode != 'breath':
246                 break
247             self.set_all_led_color(self.colorBreathR*i/self.breathSteps,
248 self.colorBreathG*i/self.breathSteps, self.colorBreathB*i/self.breathSteps)
249             #self.show()
250             time.sleep(0.03)
251         for i in range(0, self.breathSteps):
252             if self.lightMode != 'breath':
253                 break
254             self.set_all_led_color(self.colorBreathR-(self.colorBreathR*i/self.breathSteps),
255 self.colorBreathG-(self.colorBreathG*i/self.breathSteps), self.colorBreathB-
256 (self.colorBreathB*i/self.breathSteps))
257             #self.show()
```

```

258         time.sleep(0.03)
259     def policeProcessing(self):
260         while self.lightMode == 'police':
261             for i in range(0,3):
262                 self.set_all_led_color_data(0,0,255)
263                 self.show()
264                 time.sleep(0.05)
265                 self.set_all_led_color_data(0,0,0)
266                 self.show()
267                 time.sleep(0.05)
268             if self.lightMode != 'police':
269                 break
270             time.sleep(0.1)
271             for i in range(0,3):
272                 self.set_all_led_color_data(255,0,0)
273                 self.show()
274                 time.sleep(0.05)
275                 self.set_all_led_color_data(0,0,0)
276                 self.show()
277                 time.sleep(0.05)
278             time.sleep(0.1)
279
280
281     def lightChange(self):
282         if self.lightMode == 'none':
283             self.pause()
284         elif self.lightMode == 'police':
285             self.policeProcessing()
286         elif self.lightMode == 'breath':
287             self.breathProcessing()
288
289     def run(self):
290         while 1:
291             self.__flag.wait()
292             self.lightChange()
293             pass
294
295
296
297 if __name__ == '__main__':
298     import time
299     import os
300     print("spidev version is ", spidev.__version__)
301     print("spidev device as show:")
302     os.system("ls /dev/spi*")
303
304     led = Aadept_SPI_LedPixel(8, 150) # Use MOSI for /dev/spidev0 to drive the lights
305     led.start()
306     try:
307         led.breath(128, 124, 128)
308         while True:
309             time.sleep(1)
310     except KeyboardInterrupt:
311         led.pause()
312         led.led_close()
313         print("\nhe lighting animation has safely stopped")

```

```

314         sys.exit(0)
315     except Exception as e:
316         print(f"Unknown error occurred: {e}")
317         led.pause()
318         led.led_close()
319         sys.exit(1)

```

Complete code refer to [FlowingLights.py](#)

```

001  #!/usr/bin/env python3
002  # File name   : FlowingLights.py.py
003  # Website    : www.Adeept.com
004  # Author     : Adeept
005  # Date      : 2025/04/8
006  import time
007  import sys
008  import subprocess
009  from gpiozero import PWMOutputDevice as PWM
010  import threading
011  import spidev
012  import numpy
013  from numpy import sin, cos, pi
014
015  # List of basic colors
016  base_colors = [
017      (0, 255, 255),
018      (255, 0, 0),
019      (0, 255, 0),
020      (0, 0, 255),
021      (255, 255, 0),
022      (255, 0, 255),
023      (0, 128, 255),
024      (192, 192, 192),
025      (192, 192, 0),
026      (128, 128, 128),
027      (128, 0, 0),
028      (128, 128, 0),
029      (0, 128, 0),
030      (0, 128, 128)
031  ]
032
033  # Function to generate a list of color sequences
034  def generate_color_sequences():
035      color_sequences = []
036      for i in range(len(base_colors)):
037          new_sequence = base_colors[i:] + base_colors[:i]
038          color_sequences.append(new_sequence)
039      return color_sequences
040
041  # Check the Raspberry Pi model
042  def check_rpi_model():
043      _, result = run_command("cat /proc/device-tree/model |awk '{print $3}'")
044      result = result.strip()
045      if result == '3':
046          return 3

```

```

047     elif result == '4':
048         return 4
049     elif result == '5':
050         return 5
051     else:
052         return None
053
054 # Run a command and return the result
055 def run_command(cmd=""):
056     try:
057         p = subprocess.Popen(
058             cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
059         result = p.stdout.read().decode('utf-8')
060         status = p.poll()
061         return status, result
062     except Exception as e:
063         print(f"Error occurred while running the command: {e}")
064         return None, None
065
066 # Mapping function
067 def map(x, in_min, in_max, out_min, out_max):
068     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
069
070 class Adeept_SPI_LedPixel(threading.Thread):
071     def __init__(self, count=8, bright=255, sequence='GRB', bus=0, device=0, *args, **kwargs):
072         super().__init__(*args, **kwargs)
073         self.set_led_type(sequence)
074         self.set_led_count(count)
075         self.set_led_brightness(bright)
076         self.led_begin(bus, device)
077         self.lightMode = 'none'
078         self.colorBreathR = 0
079         self.colorBreathG = 0
080         self.colorBreathB = 0
081         self.breathSteps = 10
082         # self.spi_gpio_info()
083         self.set_all_led_color(0, 0, 0)
084         self.__flag = threading.Event()
085         self.__flag.clear()
086
087     def led_begin(self, bus=0, device=0):
088         self.bus = bus
089         self.device = device
090         try:
091             self.spi = spidev.SpiDev()
092             self.spi.open(self.bus, self.device)
093             self.spi.mode = 0
094             self.led_init_state = 1
095         except OSError:
096             print("Please check the configuration in /boot/firmware/config.txt.")
097             if self.bus == 0:
098                 print("You can turn on the 'SPI' in 'Interface Options' by using 'sudo raspi-config'.")
099                 print("Or make sure that 'dtparam=spi=on' is not commented, then reboot the Raspberry
100 Pi. Otherwise spi0 will not be available.")
101             else:
102                 print(f"Please add 'dtoverlay=spi{self.bus}-2cs' at the bottom of the

```

```

103 /boot/firmware/config.txt, then reboot the Raspberry Pi. Otherwise spi{self.bus} will not be
104 available.")
105     self.led_init_state = 0
106
107     def check_spi_state(self):
108         return self.led_init_state
109
110     def spi_gpio_info(self):
111         bus_info = {
112             0: "SPI0-MOSI: GPIO10(WS2812-PIN) SPI0-MISO: GPIO9 SPI0-SCLK: GPIO11 SPI0-CE0:
113 GPIO8 SPI0-CE1: GPIO7",
114             1: "SPI1-MOSI: GPIO20(WS2812-PIN) SPI1-MISO: GPIO19 SPI1-SCLK: GPIO21 SPI1-CE0:
115 GPIO18 SPI1-CE1: GPIO17 SPI2-CE1: GPIO16",
116             2: "SPI2-MOSI: GPIO41(WS2812-PIN) SPI2-MISO: GPIO40 SPI2-SCLK: GPIO42 SPI2-CE0:
117 GPIO43 SPI2-CE1: GPIO44 SPI2-CE1: GPIO45",
118             3: "SPI3-MOSI: GPIO2(WS2812-PIN) SPI3-MISO: GPIO1 SPI3-SCLK: GPIO3 SPI3-CE0:
119 GPIO0 SPI3-CE1: GPIO24",
120             4: "SPI4-MOSI: GPIO6(WS2812-PIN) SPI4-MISO: GPIO5 SPI4-SCLK: GPIO7 SPI4-CE0:
121 GPIO4 SPI4-CE1: GPIO25",
122             5: "SPI5-MOSI: GPIO14(WS2812-PIN) SPI5-MISO: GPIO13 SPI5-SCLK: GPIO15 SPI5-CE0:
123 GPIO12 SPI5-CE1: GPIO26",
124             6: "SPI6-MOSI: GPIO20(WS2812-PIN) SPI6-MISO: GPIO19 SPI6-SCLK: GPIO21 SPI6-CE0:
125 GPIO18 SPI6-CE1: GPIO27"
126         }
127         print(bus_info.get(self.bus, f"Unknown bus number: {self.bus}"))
128
129     def led_close(self):
130         try:
131             self.set_all_led_rgb([0, 0, 0])
132             self.spi.close()
133         except Exception as e:
134             print(f"Error occurred while closing the LED: {e}")
135
136     def set_led_count(self, count):
137         self.led_count = count
138         self.led_color = [0, 0, 0] * self.led_count
139         self.led_original_color = [0, 0, 0] * self.led_count
140
141     def set_led_type(self, rgb_type):
142         try:
143             led_type = ['RGB', 'RBG', 'GRB', 'GBR', 'BRG', 'BGR']
144             led_type_offset = [0x06, 0x09, 0x12, 0x21, 0x18, 0x24]
145             index = led_type.index(rgb_type)
146             self.led_red_offset = (led_type_offset[index] >> 4) & 0x03
147             self.led_green_offset = (led_type_offset[index] >> 2) & 0x03
148             self.led_blue_offset = (led_type_offset[index] >> 0) & 0x03
149             return index
150         except ValueError:
151             self.led_red_offset = 1
152             self.led_green_offset = 0
153             self.led_blue_offset = 2
154             return -1
155
156     def set_led_brightness(self, brightness):
157         self.led_brightness = brightness
158         for i in range(self.led_count):

```

```

159         self.set_led_rgb_data(i, self.led_original_color)
160
161     def set_ledpixel(self, index, r, g, b):
162         p = [0, 0, 0]
163         p[self.led_red_offset] = round(r * self.led_brightness / 255)
164         p[self.led_green_offset] = round(g * self.led_brightness / 255)
165         p[self.led_blue_offset] = round(b * self.led_brightness / 255)
166         self.led_original_color[index * 3 + self.led_red_offset] = r
167         self.led_original_color[index * 3 + self.led_green_offset] = g
168         self.led_original_color[index * 3 + self.led_blue_offset] = b
169         for i in range(3):
170             self.led_color[index * 3 + i] = p[i]
171
172     def setSomeColor_data(self, index, r, g, b):
173         self.set_ledpixel(index, r, g, b)
174
175     def set_led_rgb_data(self, index, color):
176         self.set_ledpixel(index, color[0], color[1], color[2])
177
178     def setSomeColor(self, index, r, g, b):
179         self.set_ledpixel(index, r, g, b)
180         self.show()
181
182     def set_led_rgb(self, index, color):
183         self.set_led_rgb_data(index, color)
184         self.show()
185
186     def set_all_led_color_data(self, r, g, b):
187         for i in range(self.led_count):
188             self.setSomeColor_data(i, r, g, b)
189
190     def set_all_led_rgb_data(self, color):
191         for i in range(self.led_count):
192             self.set_led_rgb_data(i, color)
193
194     def set_all_led_color(self, r, g, b):
195         for i in range(self.led_count):
196             self.setSomeColor_data(i, r, g, b)
197         self.show()
198
199     def set_all_led_rgb(self, color):
200         for i in range(self.led_count):
201             self.set_led_rgb_data(i, color)
202         self.show()
203
204     def write_ws2812_numpy8(self):
205         d = numpy.array(self.led_color).ravel()
206         tx = numpy.zeros(len(d) * 8, dtype=numpy.uint8)
207         for ibit in range(8):
208             tx[7 - ibit::8] = ((d >> ibit) & 1) * 0x78 + 0x80
209         if self.led_init_state != 0:
210             if self.bus == 0:
211                 self.spi.xfer(tx.tolist(), int(8 / 1.25e-6))
212             else:
213                 self.spi.xfer(tx.tolist(), int(8 / 1.0e-6))
214

```

```

215     def write_ws2812_numpy4(self):
216         d = numpy.array(self.led_color).ravel()
217         tx = numpy.zeros(len(d) * 4, dtype=numpy.uint8)
218         for ibit in range(4):
219             tx[3 - ibit::4] = ((d >> (2 * ibit + 1)) & 1) * 0x60 + ((d >> (2 * ibit + 0)) & 1) * 0x06 +
220 0x88
221         if self.led_init_state != 0:
222             if self.bus == 0:
223                 self.spi.xfer(tx.tolist(), int(4 / 1.25e-6))
224             else:
225                 self.spi.xfer(tx.tolist(), int(4 / 1.0e-6))
226
227     def show(self, mode=1):
228         if mode == 1:
229             write_ws2812 = self.write_ws2812_numpy8
230         else:
231             write_ws2812 = self.write_ws2812_numpy4
232         write_ws2812()
233
234     def wheel(self, pos):
235         if pos < 85:
236             return [(255 - pos * 3), (pos * 3), 0]
237         elif pos < 170:
238             pos = pos - 85
239             return [0, (255 - pos * 3), (pos * 3)]
240         else:
241             pos = pos - 170
242             return [(pos * 3), 0, (255 - pos * 3)]
243
244     def hsv2rgb(self, h, s, v):
245         h = h % 360
246         rgb_max = round(v * 2.55)
247         rgb_min = round(rgb_max * (100 - s) / 100)
248         i = round(h / 60)
249         diff = round(h % 60)
250         rgb_adj = round((rgb_max - rgb_min) * diff / 60)
251         if i == 0:
252             r = rgb_max
253             g = rgb_min + rgb_adj
254             b = rgb_min
255         elif i == 1:
256             r = rgb_max - rgb_adj
257             g = rgb_max
258             b = rgb_min
259         elif i == 2:
260             r = rgb_min
261             g = rgb_max
262             b = rgb_min + rgb_adj
263         elif i == 3:
264             r = rgb_min
265             g = rgb_max - rgb_adj
266             b = rgb_max
267         elif i == 4:
268             r = rgb_min + rgb_adj
269             g = rgb_min
270             b = rgb_max

```



```

271         else:
272             r = rgb_max
273             g = rgb_min
274             b = rgb_max - rgb_adj
275         return [r, g, b]
276
277     def police(self):
278         self.lightMode = 'police'
279         self.resume()
280
281     def breath(self, R_input, G_input, B_input):
282         self.lightMode = 'breath'
283         self.colorBreathR = R_input
284         self.colorBreathG = G_input
285         self.colorBreathB = B_input
286         self.resume()
287
288     def resume(self):
289         self.__flag.set()
290
291     def pause(self):
292         self.lightMode = 'none'
293         self.set_all_led_color_data(0, 0, 0)
294         self.__flag.clear()
295
296     def breathProcessing(self):
297         while self.lightMode == 'breath':
298             for i in range(0, self.breathSteps):
299                 if self.lightMode != 'breath':
300                     break
301                 self.set_all_led_color(self.colorBreathR * i / self.breathSteps,
302                                     self.colorBreathG * i / self.breathSteps,
303                                     self.colorBreathB * i / self.breathSteps)
304                 time.sleep(0.03)
305             for i in range(0, self.breathSteps):
306                 if self.lightMode != 'breath':
307                     break
308                 self.set_all_led_color(self.colorBreathR - (self.colorBreathR * i / self.breathSteps),
309                                     self.colorBreathG - (self.colorBreathG * i / self.breathSteps),
310                                     self.colorBreathB - (self.colorBreathB * i / self.breathSteps))
311                 time.sleep(0.03)
312
313     def policeProcessing(self):
314         while self.lightMode == 'police':
315             for i in range(0, 3):
316                 self.set_all_led_color_data(0, 0, 255)
317                 self.show()
318                 time.sleep(0.05)
319                 self.set_all_led_color_data(0, 0, 0)
320                 self.show()
321                 time.sleep(0.05)
322             if self.lightMode != 'police':
323                 break
324             time.sleep(0.1)
325             for i in range(0, 3):
326                 self.set_all_led_color_data(255, 0, 0)

```

```

327         self.show()
328         time.sleep(0.05)
329         self.set_all_led_color_data(0, 0, 0)
330         self.show()
331         time.sleep(0.05)
332         time.sleep(0.1)
333
334     def setDifferentColors(self, colors):
335         max_led = min(len(colors), self.led_count)
336         for i in range(max_led):
337             r, g, b = colors[i]
338             self.setSomeColor_data(i, r, g, b)
339         self.show()
340
341     def lightChange(self):
342         if self.lightMode == 'none':
343             self.pause()
344         elif self.lightMode == 'police':
345             self.policeProcessing()
346         elif self.lightMode == 'breath':
347             self.breathProcessing()
348
349     def run(self):
350         while True:
351             self.__flag.wait()
352             self.lightChange()
353
354 if __name__ == '__main__':
355     # Configuration parameters
356     LED_COUNT = 14
357     BRIGHTNESS = 50
358     ANIMATION_DELAY = 0.3
359
360     #Generate a color sequence list
361     color_sequences = generate_color_sequences()
362
363     # Initialize LED controller
364     RL = Adeept_SPI_LedPixel(count=LED_COUNT, bright=BRIGHTNESS)
365     RL.start()
366
367     try:
368         while True:
369             for sequence in color_sequences:
370                 RL.setDifferentColors(sequence)
371                 time.sleep(ANIMATION_DELAY)
372     except KeyboardInterrupt:
373         # Safe shutdown process
374         RL.pause()
375         RL.led_close()
376         print("\nhe lighting animation has safely stopped")
377         sys.exit(0)
378     except Exception as e:
379         print(f"Unknown error occurred: {e}")
380         RL.pause()
381         RL.led_close()
382         sys.exit(1)

```

Code explanation

BreathingLight.py

Initialization Stage:

The code starts by initializing the necessary libraries, such as spidev for SPI communication and any custom - made LED control libraries like Adeept_SPI_LedPixel. It also initializes the SPI connection, sets the appropriate mode and speed for data transfer.

Loop Control Process:

A loop is used to create the breathing effect. This loop gradually increases or decreases the RGB values of the LEDs over time. For example, it could use a sine - wave - like function to smoothly change the brightness, making the LEDs appear to breathe.

The code includes proper resource management. When the program is terminated (e.g., by pressing **Ctrl + C**), it closes the SPI connection and releases any other resources used during the execution.

FlowingLights.py

Initialization Stage:

The code starts by initializing the necessary libraries, such as spidev for SPI communication and any custom - made LED control libraries like Adeept_SPI_LedPixel. It also initializes the SPI connection, sets the appropriate mode and speed for data transfer.

Loop Control Process:

A loop is used to control the movement of the flowing lights. It updates the state of the LEDs based on the defined sequence, moving the "flow" of light across the LED strip.

The code includes proper resource management. When the program is terminated (e.g., by pressing **Ctrl + C**), it closes the SPI connection and releases any other resources used during the execution.