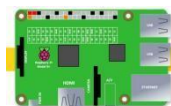
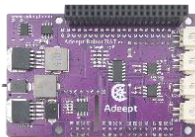




Lesson 11 Light up the WS2812

11.1 Overview

This lesson focuses on teaching how to control WS2812 RGB LEDs using a Raspberry Pi with the Adept 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.

11.2 Required Components

Components	Quantity	Picture
Raspberry Pi	1	
Adept Robot HAT V3.2	1	
3 pin cable	1	
WS2812 RGB LED	1	

11.3 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.

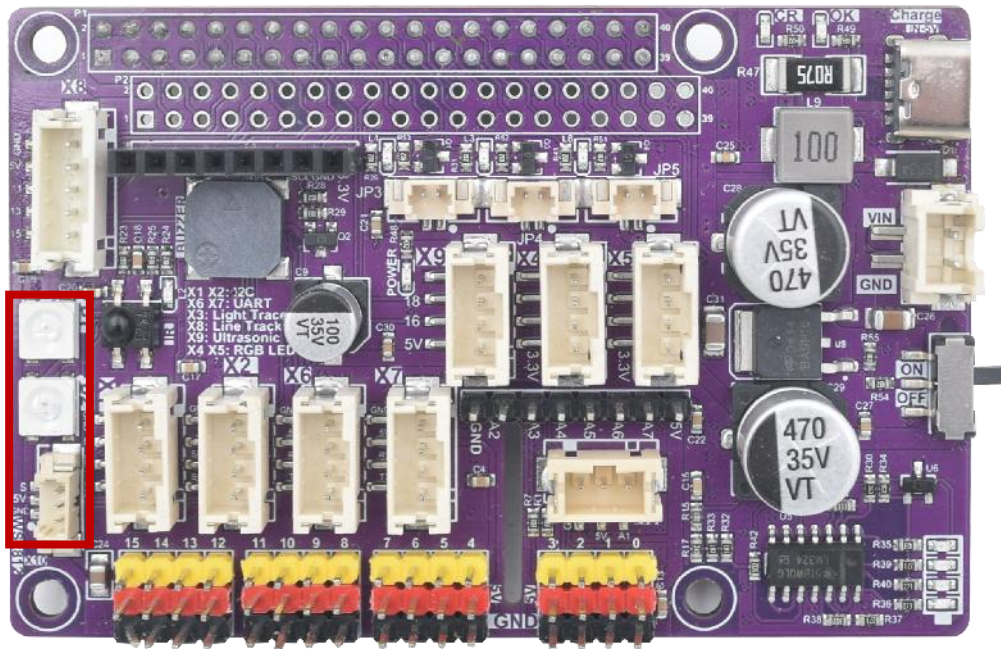
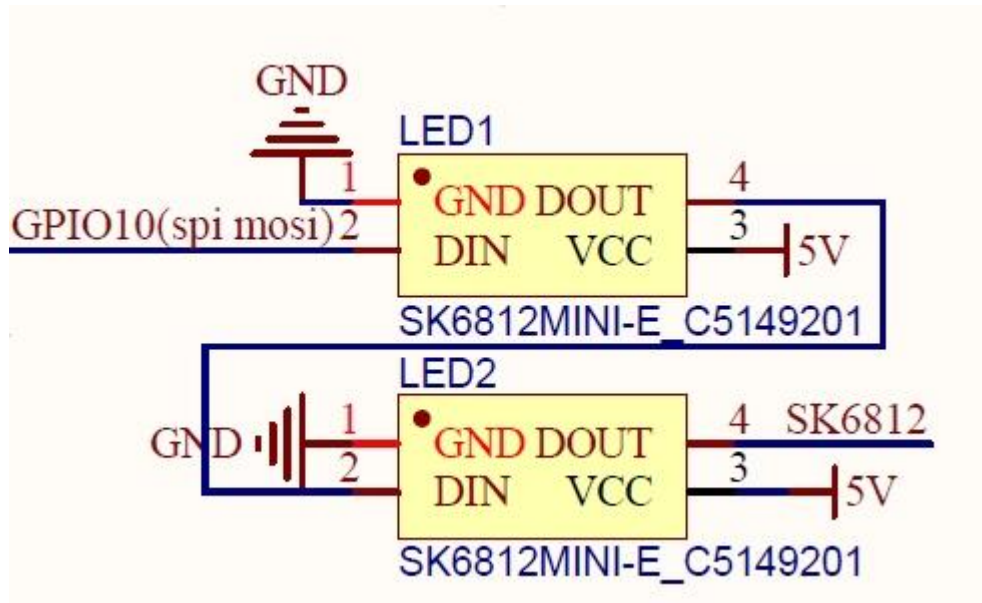
PINS of Raspberry Pi	Sensor
GPIO10	WS2812

11.4 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.

Adeept 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, ...).



11.5 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_4WD_Smart_Car_for_RPi/Examples/05_WS2812/
```

```
pi@raspberrypi:~ $ cd Adeept_4WD_Smart_Car_for_RPi/Examples/05_WS2812/  
pi@raspberrypi:~/Adeeps_4WD_Smart_Car_for_RPi/Examples/05_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:~/Adeeps_4WD_Smart_Car_for_RPi/Examples/05_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:~/Adeeps_4WD_Smart_Car_for_RPi/Examples/05_WS2812 $ sudo python3 BreathingLight.py  
spidev version is 3.5  
spidev device as show:  
/dev/spidev0.0 /dev/spidev0.1
```

5. **Observation and Termination:** The program prints the version of the spidev library and available SPI devices, creates an instance of the Adeeps_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 BreathingLight.py program:

```
sudo python3 FlowingLights.py
```

```
pi@raspberrypi:~/Adeept_4WD_Smart_Car_for_RPi/Examples/05_WS2812 $ sudo python3 FlowingLights.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.

11.6 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      : 2025/03/5
006 import spidev
007 import threading
008 import numpy
009 from numpy import sin, cos, pi
010 import time
011 class Adeept_SPI_LedPixel(threading.Thread):
012     def __init__(self, count = 8, bright = 255, sequence='GRB', bus = 0, device = 0, *args, **kwargs):
013         self.set_led_type(sequence)
014         self.set_led_count(count)
015         self.set_led_brightness(bright)
016         self.led_begin(bus, device)
017         self.lightMode = 'none'
018         self.colorBreathR = 0
019         self.colorBreathG = 0
020         self.colorBreathB = 0
021         self.breathSteps = 10
022         self.set_all_led_color(0,0,0)
023         super(Adeept_SPI_LedPixel, self).__init__(*args, **kwargs)
024         self.__flag = threading.Event()
025         self.__flag.clear()
026     def led_begin(self, bus = 0, device = 0):
027         self.bus = bus
028         self.device = device
029         try:
030             self.spi = spidev.SpiDev()
031             self.spi.open(self.bus, self.device)
032             self.spi.mode = 0
033             self.led_init_state = 1
034         except OSError:
035             print("Please check the configuration in /boot/firmware/config.txt.")
036             if self.bus == 0:
```

```

037         print("You can turn on the 'SPI' in 'Interface Options' by using 'sudo raspi-config'.")
038         print("Or make sure that 'dtoverlay=spi=on' is not commented, then reboot the Raspberry
039 Pi. Otherwise spi0 will not be available.")
040     else:
041         print("Please add 'dtoverlay=spi{}-2cs' at the bottom of the /boot/firmware/config.txt,
042 then reboot the Raspberry Pi. otherwise spi{} will not be available.".format(self.bus, self.bus))
043         self.led_init_state = 0
044
045     def check_spi_state(self):
046         return self.led_init_state
047
048     def spi_gpio_info(self):
049         if self.bus == 0:
050             print("SPI0-MOSI: GPIO10(W2812-PIN) SPI0-MISO: GPIO9 SPI0-SCLK: GPIO11 SPI0-CE0:
051 GPIO8 SPI0-CE1: GPIO7")
052         elif self.bus == 1:
053             print("SPI1-MOSI: GPIO20(W2812-PIN) SPI1-MISO: GPIO19 SPI1-SCLK: GPIO21 SPI1-CE0:
054 GPIO18 SPI1-CE1: GPIO17 SPI0-CE1: GPIO16")
055         elif self.bus == 2:
056             print("SPI2-MOSI: GPIO41(W2812-PIN) SPI2-MISO: GPIO40 SPI2-SCLK: GPIO42 SPI2-CE0:
057 GPIO43 SPI2-CE1: GPIO44 SPI2-CE1: GPIO45")
058         elif self.bus == 3:
059             print("SPI3-MOSI: GPIO2(W2812-PIN) SPI3-MISO: GPIO1 SPI3-SCLK: GPIO3 SPI3-CE0:
060 GPIO0 SPI3-CE1: GPIO24")
061         elif self.bus == 4:
062             print("SPI4-MOSI: GPIO6(W2812-PIN) SPI4-MISO: GPIO5 SPI4-SCLK: GPIO7 SPI4-CE0:
063 GPIO4 SPI4-CE1: GPIO25")
064         elif self.bus == 5:
065             print("SPI5-MOSI: GPIO14(W2812-PIN) SPI5-MISO: GPIO13 SPI5-SCLK: GPIO15 SPI5-CE0:
066 GPIO12 SPI5-CE1: GPIO26")
067         elif self.bus == 6:
068             print("SPI6-MOSI: GPIO20(W2812-PIN) SPI6-MISO: GPIO19 SPI6-SCLK: GPIO21 SPI6-CE0:
069 GPIO18 SPI6-CE1: GPIO27")
070
071     def led_close(self):
072         self.set_all_led_rgb([0,0,0])
073         self.spi.close()
074
075     def set_led_count(self, count):
076         self.led_count = count
077         self.led_color = [0,0,0] * self.led_count
078         self.led_original_color = [0,0,0] * self.led_count
079
080     def set_led_type(self, rgb_type):
081         try:
082             led_type = ['RGB', 'RBG', 'GRB', 'GBR', 'BRG', 'BGR']
083             led_type_offset = [0x06, 0x09, 0x12, 0x21, 0x18, 0x24]
084             index = led_type.index(rgb_type)
085             self.led_red_offset = (led_type_offset[index]>>4) & 0x03
086             self.led_green_offset = (led_type_offset[index]>>2) & 0x03
087             self.led_blue_offset = (led_type_offset[index]>>0) & 0x03
088             return index
089         except ValueError:
090             self.led_red_offset = 1
091             self.led_green_offset = 0
092             self.led_blue_offset = 2

```

```

093         return -1
094
095     def set_led_brightness(self, brightness):
096         self.led_brightness = brightness
097         for i in range(self.led_count):
098             self.set_led_rgb_data(i, self.led_original_color)
099
100     def set_ledpixel(self, index, r, g, b):
101         p = [0,0,0]
102         p[self.led_red_offset] = round(r * self.led_brightness / 255)
103         p[self.led_green_offset] = round(g * self.led_brightness / 255)
104         p[self.led_blue_offset] = round(b * self.led_brightness / 255)
105         self.led_original_color[index*3+self.led_red_offset] = r
106         self.led_original_color[index*3+self.led_green_offset] = g
107         self.led_original_color[index*3+self.led_blue_offset] = b
108         for i in range(3):
109             self.led_color[index*3+i] = p[i]
110
111     def set_led_color_data(self, index, r, g, b):
112         self.set_ledpixel(index, r, g, b)
113
114     def set_led_rgb_data(self, index, color):
115         self.set_ledpixel(index, color[0], color[1], color[2])
116
117     def set_led_color(self, index, r, g, b):
118         self.set_ledpixel(index, r, g, b)
119         self.show()
120
121     def set_led_rgb(self, index, color):
122         self.set_led_rgb_data(index, color)
123         self.show()
124
125     def set_all_led_color_data(self, r, g, b):
126         for i in range(self.led_count):
127             self.set_led_color_data(i, r, g, b)
128
129     def set_all_led_rgb_data(self, color):
130         for i in range(self.led_count):
131             self.set_led_rgb_data(i, color)
132
133     def set_all_led_color(self, r, g, b):
134         for i in range(self.led_count):
135             self.set_led_color_data(i, r, g, b)
136         self.show()
137
138     def set_all_led_rgb(self, color):
139         for i in range(self.led_count):
140             self.set_led_rgb_data(i, color)
141         self.show()
142
143     def write_ws2812_numpy8(self):
144         d = numpy.array(self.led_color).ravel() #Converts data into a one-dimensional array
145         tx = numpy.zeros(len(d)*8, dtype=numpy.uint8) #Each RGB color has 8 bits, each represented by
146         a uint8 type data
147         for ibit in range(8): #Convert each bit of data to the data that the
148             spi will send

```

```

149         tx[7-ibit::8]=((d>>ibit)&1)*0x78 + 0x80      #T0H=1,T0L=7, T1H=5,T1L=3   #0b11111000 mean
150 T1(0.78125us), 0b10000000 mean T0(0.15625us)
151         if self.led_init_state != 0:
152             if self.bus == 0:
153                 self.spi.xfer(tx.tolist(), int(8/1.25e-6))          #Send color data at a frequency of
154 6.4Mhz
155             else:
156                 self.spi.xfer(tx.tolist(), int(8/1.0e-6))          #Send color data at a frequency of
157 8Mhz
158
159     def write_ws2812_numpy4(self):
160         d=numpy.array(self.led_color).ravel()
161         tx=numpy.zeros(len(d)*4, dtype=numpy.uint8)
162         for ibit in range(4):
163             tx[3-ibit::4]=((d>>(2*ibit+1))&1)*0x60 + ((d>>(2*ibit+0))&1)*0x06 + 0x88
164         if self.led_init_state != 0:
165             if self.bus == 0:
166                 self.spi.xfer(tx.tolist(), int(4/1.25e-6))
167             else:
168                 self.spi.xfer(tx.tolist(), int(4/1.0e-6))
169
170     def show(self, mode = 1):
171         if mode == 1:
172             write_ws2812 = self.write_ws2812_numpy8
173         else:
174             write_ws2812 = self.write_ws2812_numpy4
175         write_ws2812()
176
177     def wheel(self, pos):
178         if pos < 85:
179             return [(255 - pos * 3), (pos * 3), 0]
180         elif pos < 170:
181             pos = pos - 85
182             return [0, (255 - pos * 3), (pos * 3)]
183         else:
184             pos = pos - 170
185             return [(pos * 3), 0, (255 - pos * 3)]
186
187     def hsv2rgb(self, h, s, v):
188         h = h % 360
189         rgb_max = round(v * 2.55)
190         rgb_min = round(rgb_max * (100 - s) / 100)
191         i = round(h / 60)
192         diff = round(h % 60)
193         rgb_adj = round((rgb_max - rgb_min) * diff / 60)
194         if i == 0:
195             r = rgb_max
196             g = rgb_min + rgb_adj
197             b = rgb_min
198         elif i == 1:
199             r = rgb_max - rgb_adj
200             g = rgb_max
201             b = rgb_min
202         elif i == 2:
203             r = rgb_min
204             g = rgb_max

```

```
205         b = rgb_min + rgb_adj
206     elif i == 3:
207         r = rgb_min
208         g = rgb_max - rgb_adj
209         b = rgb_max
210     elif i == 4:
211         r = rgb_min + rgb_adj
212         g = rgb_min
213         b = rgb_max
214     else:
215         r = rgb_max
216         g = rgb_min
217         b = rgb_max - rgb_adj
218     return [r, g, b]
219
220     def police(self):
221         self.lightMode = 'police'
222         self.resume()
223
224     def breath(self, R_input, G_input, B_input):
225         self.lightMode = 'breath'
226         self.colorBreathR = R_input
227         self.colorBreathG = G_input
228         self.colorBreathB = B_input
229         self.resume()
230
231     def resume(self):
232         self.__flag.set()
233
234
235     def breathProcessing(self):
236         while self.lightMode == 'breath':
237             for i in range(0,self.breathSteps):
238                 if self.lightMode != 'breath':
239                     break
240                 self.set_all_led_color(self.colorBreathR*i/self.breathSteps,
241 self.colorBreathG*i/self.breathSteps, self.colorBreathB*i/self.breathSteps)
242                 #self.show()
243                 time.sleep(0.03)
244             for i in range(0,self.breathSteps):
245                 if self.lightMode != 'breath':
246                     break
247                 self.set_all_led_color(self.colorBreathR-(self.colorBreathR*i/self.breathSteps),
248 self.colorBreathG-(self.colorBreathG*i/self.breathSteps), self.colorBreathB-
249 (self.colorBreathB*i/self.breathSteps))
250                 #self.show()
251                 time.sleep(0.03)
252     def policeProcessing(self):
253         while self.lightMode == 'police':
254             for i in range(0,3):
255                 self.set_all_led_color_data(0,0,255)
256                 self.show()
257                 time.sleep(0.05)
258                 self.set_all_led_color_data(0,0,0)
259                 self.show()
260                 time.sleep(0.05)
```

```

261         if self.lightMode != 'police':
262             break
263         time.sleep(0.1)
264         for i in range(0,3):
265             self.set_all_led_color_data(255,0,0)
266             self.show()
267             time.sleep(0.05)
268             self.set_all_led_color_data(0,0,0)
269             self.show()
270             time.sleep(0.05)
271         time.sleep(0.1)
272
273
274     def lightChange(self):
275         if self.lightMode == 'none':
276             self.pause()
277         elif self.lightMode == 'police':
278             self.policeProcessing()
279         elif self.lightMode == 'breath':
280             self.breathProcessing()
281
282     def run(self):
283         while 1:
284             self.__flag.wait()
285             self.lightChange()
286             pass
287
288
289 if __name__ == '__main__':
290     import time
291     import os
292     print("spidev version is ", spidev.__version__)
293     print("spidev device as show:")
294     os.system("ls /dev/spi*")
295
296     led = Adeept_SPI_LedPixel(5, 150) # Use MOSI for /dev/spidev0 to drive the lights
297     led.start()
298     try:
299         led.breath(128, 124, 128)
300         while True:
301             time.sleep(1)
302     except KeyboardInterrupt:
303         led.pause()
304         led.led_close()

```

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

```

001 #!/usr/bin/env python3
002 # File name   : servo.py
003 # Description : Control lights
004 # Author      : William
005 # Date        : 2019/02/23

```

```

006 import time
007 import sys
008 import subprocess
009 from rpi_ws281x import *
010 import threading
011 import spidev
012 import numpy
013
014 # List of basic colors
015 base_colors = [
016     (0, 255, 255),
017     (255, 0, 0),
018     (0, 255, 0),
019     (0, 0, 255),
020     (255, 255, 0),
021     (255, 0, 255),
022     (0, 128, 255),
023     (192, 192, 192),
024     (192, 192, 0),
025     (128, 128, 128),
026     (128, 0, 0),
027     (128, 128, 0),
028     (0, 128, 0),
029     (0, 128, 128)
030 ]
031
032 # Function to generate a list of color sequences
033 def generate_color_sequences():
034     color_sequences = []
035     for i in range(len(base_colors)):
036         new_sequence = base_colors[i:] + base_colors[:i]
037         color_sequences.append(new_sequence)
038     return color_sequences
039
040 # Check the Raspberry Pi model
041 def check_rpi_model():
042     _, result = run_command("cat /proc/device-tree/model |awk '{print $3}'")
043     result = result.strip()
044     if result == '3':
045         return 3
046     elif result == '4':
047         return 4
048     elif result == '5':
049         return 5
050     else:
051         return None
052
053 # Run a command and return the result
054 def run_command(cmd=""):
055     try:
056         p = subprocess.Popen(
057             cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
058         result = p.stdout.read().decode('utf-8')
059         status = p.poll()
060         return status, result
061     except Exception as e:

```

```

062         print(f"Error occurred while running the command: {e}")
063         return None, None
064
065     # Mapping function
066     def map(x, in_min, in_max, out_min, out_max):
067         return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
068
069     class Adeept_SPI_LedPixel(threading.Thread):
070         def __init__(self, count=8, bright=255, sequence='GRB', bus=0, device=0, *args, **kwargs):
071             super().__init__(*args, **kwargs)
072             self.set_led_type(sequence)
073             self.set_led_count(count)
074             self.set_led_brightness(bright)
075             self.led_begin(bus, device)
076             self.lightMode = 'none'
077             self.colorBreathR = 0
078             self.colorBreathG = 0
079             self.colorBreathB = 0
080             self.breathSteps = 10
081             # self.spi_gpio_info()
082             self.set_all_led_color(0, 0, 0)
083             self.__flag = threading.Event()
084             self.__flag.clear()
085
086         def led_begin(self, bus=0, device=0):
087             self.bus = bus
088             self.device = device
089             try:
090                 self.spi = spidev.SpiDev()
091                 self.spi.open(self.bus, self.device)
092                 self.spi.mode = 0
093                 self.led_init_state = 1
094             except OSError:
095                 print("Please check the configuration in /boot/firmware/config.txt.")
096                 if self.bus == 0:
097                     print("You can turn on the 'SPI' in 'Interface Options' by using 'sudo raspi-config'.")
098                     print("Or make sure that 'dtparam=spi=on' is not commented, then reboot the Raspberry
099 Pi. Otherwise spi0 will not be available.")
100                 else:
101                     print(f"Please add 'dtoverlay=spi{self.bus}-2cs' at the bottom of the
102 /boot/firmware/config.txt, then reboot the Raspberry Pi. Otherwise spi{self.bus} will not be
103 available.")
104                 self.led_init_state = 0
105
106         def check_spi_state(self):
107             return self.led_init_state
108
109         def spi_gpio_info(self):
110             bus_info = {
111                 0: "SPI0-MOSI: GPIO10(W2812-PIN)  SPI0-MISO: GPIO9  SPI0-SCLK: GPIO11  SPI0-CE0:
112 GPIO8  SPI0-CE1: GPIO7",
113                 1: "SPI1-MOSI: GPIO20(W2812-PIN)  SPI1-MISO: GPIO19  SPI1-SCLK: GPIO21  SPI1-CE0:
114 GPIO18  SPI1-CE1: GPIO17  SPI0-CE1: GPIO16",
115                 2: "SPI2-MOSI: GPIO41(W2812-PIN)  SPI2-MISO: GPIO40  SPI2-SCLK: GPIO42  SPI2-CE0:
116 GPIO43  SPI2-CE1: GPIO44  SPI2-CE1: GPIO45",
117                 3: "SPI3-MOSI: GPIO2(W2812-PIN)  SPI3-MISO: GPIO1  SPI3-SCLK: GPIO3  SPI3-CE0:

```

```

118 GPIO0 SPI3-CE1: GPIO24",
119         4: "SPI4-MOSI: GPIO6(W2812-PIN) SPI4-MISO: GPIO5 SPI4-SCLK: GPIO7 SPI4-CE0:
120 GPIO4 SPI4-CE1: GPIO25",
121         5: "SPI5-MOSI: GPIO14(W2812-PIN) SPI5-MISO: GPIO13 SPI5-SCLK: GPIO15 SPI5-CE0:
122 GPIO12 SPI5-CE1: GPIO26",
123         6: "SPI6-MOSI: GPIO20(W2812-PIN) SPI6-MISO: GPIO19 SPI6-SCLK: GPIO21 SPI6-CE0:
124 GPIO18 SPI6-CE1: GPIO27"
125     }
126     print(bus_info.get(self.bus, f"Unknown bus number: {self.bus}"))
127
128     def led_close(self):
129         try:
130             self.set_all_led_rgb([0, 0, 0])
131             self.spi.close()
132         except Exception as e:
133             print(f"Error occurred while closing the LED: {e}")
134
135     def set_led_count(self, count):
136         self.led_count = count
137         self.led_color = [0, 0, 0] * self.led_count
138         self.led_original_color = [0, 0, 0] * self.led_count
139
140     def set_led_type(self, rgb_type):
141         try:
142             led_type = ['RGB', 'RBG', 'GRB', 'GBR', 'BRG', 'BGR']
143             led_type_offset = [0x06, 0x09, 0x12, 0x21, 0x18, 0x24]
144             index = led_type.index(rgb_type)
145             self.led_red_offset = (led_type_offset[index] >> 4) & 0x03
146             self.led_green_offset = (led_type_offset[index] >> 2) & 0x03
147             self.led_blue_offset = (led_type_offset[index] >> 0) & 0x03
148             return index
149         except ValueError:
150             self.led_red_offset = 1
151             self.led_green_offset = 0
152             self.led_blue_offset = 2
153             return -1
154
155     def set_led_brightness(self, brightness):
156         self.led_brightness = brightness
157         for i in range(self.led_count):
158             self.set_led_rgb_data(i, self.led_original_color)
159
160     def set_ledpixel(self, index, r, g, b):
161         p = [0, 0, 0]
162         p[self.led_red_offset] = round(r * self.led_brightness / 255)
163         p[self.led_green_offset] = round(g * self.led_brightness / 255)
164         p[self.led_blue_offset] = round(b * self.led_brightness / 255)
165         self.led_original_color[index * 3 + self.led_red_offset] = r
166         self.led_original_color[index * 3 + self.led_green_offset] = g
167         self.led_original_color[index * 3 + self.led_blue_offset] = b
168         for i in range(3):
169             self.led_color[index * 3 + i] = p[i]
170
171     def setSomeColor_data(self, index, r, g, b):
172         self.set_ledpixel(index, r, g, b)
173

```

```

174     def set_led_rgb_data(self, index, color):
175         self.set_ledpixel(index, color[0], color[1], color[2])
176
177     def setSomeColor(self, index, r, g, b):
178         self.set_ledpixel(index, r, g, b)
179         self.show()
180
181     def set_led_rgb(self, index, color):
182         self.set_led_rgb_data(index, color)
183         self.show()
184
185     def set_all_led_color_data(self, r, g, b):
186         for i in range(self.led_count):
187             self.setSomeColor_data(i, r, g, b)
188
189     def set_all_led_rgb_data(self, color):
190         for i in range(self.led_count):
191             self.set_led_rgb_data(i, color)
192
193     def set_all_led_color(self, r, g, b):
194         for i in range(self.led_count):
195             self.setSomeColor_data(i, r, g, b)
196         self.show()
197
198     def set_all_led_rgb(self, color):
199         for i in range(self.led_count):
200             self.set_led_rgb_data(i, color)
201         self.show()
202
203     def write_ws2812_numpy8(self):
204         d = numpy.array(self.led_color).ravel()
205         tx = numpy.zeros(len(d) * 8, dtype=numpy.uint8)
206         for ibit in range(8):
207             tx[7 - ibit::8] = ((d >> ibit) & 1) * 0x78 + 0x80
208         if self.led_init_state != 0:
209             if self.bus == 0:
210                 self.spi.xfer(tx.tolist(), int(8 / 1.25e-6))
211             else:
212                 self.spi.xfer(tx.tolist(), int(8 / 1.0e-6))
213
214     def write_ws2812_numpy4(self):
215         d = numpy.array(self.led_color).ravel()
216         tx = numpy.zeros(len(d) * 4, dtype=numpy.uint8)
217         for ibit in range(4):
218             tx[3 - ibit::4] = ((d >> (2 * ibit + 1)) & 1) * 0x60 + ((d >> (2 * ibit + 0)) & 1) * 0x06 +
219 0x88
220         if self.led_init_state != 0:
221             if self.bus == 0:
222                 self.spi.xfer(tx.tolist(), int(4 / 1.25e-6))
223             else:
224                 self.spi.xfer(tx.tolist(), int(4 / 1.0e-6))
225
226     def show(self, mode=1):
227         if mode == 1:
228             write_ws2812 = self.write_ws2812_numpy8
229         else:

```

```

230         write_ws2812 = self.write_ws2812_numpy4
231         write_ws2812()
232
233     def wheel(self, pos):
234         if pos < 85:
235             return [(255 - pos * 3), (pos * 3), 0]
236         elif pos < 170:
237             pos = pos - 85
238             return [0, (255 - pos * 3), (pos * 3)]
239         else:
240             pos = pos - 170
241             return [(pos * 3), 0, (255 - pos * 3)]
242
243     def hsv2rgb(self, h, s, v):
244         h = h % 360
245         rgb_max = round(v * 2.55)
246         rgb_min = round(rgb_max * (100 - s) / 100)
247         i = round(h / 60)
248         diff = round(h % 60)
249         rgb_adj = round((rgb_max - rgb_min) * diff / 60)
250         if i == 0:
251             r = rgb_max
252             g = rgb_min + rgb_adj
253             b = rgb_min
254         elif i == 1:
255             r = rgb_max - rgb_adj
256             g = rgb_max
257             b = rgb_min
258         elif i == 2:
259             r = rgb_min
260             g = rgb_max
261             b = rgb_min + rgb_adj
262         elif i == 3:
263             r = rgb_min
264             g = rgb_max - rgb_adj
265             b = rgb_max
266         elif i == 4:
267             r = rgb_min + rgb_adj
268             g = rgb_min
269             b = rgb_max
270         else:
271             r = rgb_max
272             g = rgb_min
273             b = rgb_max - rgb_adj
274         return [r, g, b]
275
276     def police(self):
277         self.lightMode = 'police'
278         self.resume()
279
280     def breath(self, R_input, G_input, B_input):
281         self.lightMode = 'breath'
282         self.colorBreathR = R_input
283         self.colorBreathG = G_input
284         self.colorBreathB = B_input
285         self.resume()

```

```

286
287     def resume(self):
288         self.__flag.set()
289
290     def pause(self):
291         self.lightMode = 'none'
292         self.set_all_led_color_data(0, 0, 0)
293         self.__flag.clear()
294
295     def breathProcessing(self):
296         while self.lightMode == 'breath':
297             for i in range(0, self.breathSteps):
298                 if self.lightMode != 'breath':
299                     break
300                 self.set_all_led_color(self.colorBreathR * i / self.breathSteps,
301                                     self.colorBreathG * i / self.breathSteps,
302                                     self.colorBreathB * i / self.breathSteps)
303                 time.sleep(0.03)
304             for i in range(0, self.breathSteps):
305                 if self.lightMode != 'breath':
306                     break
307                 self.set_all_led_color(self.colorBreathR - (self.colorBreathR * i / self.breathSteps),
308                                     self.colorBreathG - (self.colorBreathG * i / self.breathSteps),
309                                     self.colorBreathB - (self.colorBreathB * i / self.breathSteps))
310                 time.sleep(0.03)
311
312     def policeProcessing(self):
313         while self.lightMode == 'police':
314             for i in range(0, 3):
315                 self.set_all_led_color_data(0, 0, 255)
316                 self.show()
317                 time.sleep(0.05)
318                 self.set_all_led_color_data(0, 0, 0)
319                 self.show()
320                 time.sleep(0.05)
321             if self.lightMode != 'police':
322                 break
323             time.sleep(0.1)
324             for i in range(0, 3):
325                 self.set_all_led_color_data(255, 0, 0)
326                 self.show()
327                 time.sleep(0.05)
328                 self.set_all_led_color_data(0, 0, 0)
329                 self.show()
330                 time.sleep(0.05)
331             time.sleep(0.1)
332
333     def setDifferentColors(self, colors):
334         max_led = min(len(colors), self.led_count)
335         for i in range(max_led):
336             r, g, b = colors[i]
337             self.setSomeColor_data(i, r, g, b)
338         self.show()
339
340     def lightChange(self):
341         if self.lightMode == 'none':

```

```

342         self.pause()
343     elif self.lightMode == 'police':
344         self.policeProcessing()
345     elif self.lightMode == 'breath':
346         self.breathProcessing()
347
348     def run(self):
349         while True:
350             self.__flag.wait()
351             self.lightChange()
352
353 if __name__ == '__main__':
354     # Configuration parameters
355     LED_COUNT = 5
356     BRIGHTNESS = 50
357     ANIMATION_DELAY = 0.3
358
359     #Generate a color sequence list
360     color_sequences = generate_color_sequences()
361
362     # Initialize LED controller
363     RL = Adeept_SPI_LedPixel(count=LED_COUNT, bright=BRIGHTNESS)
364     RL.start()
365
366     try:
367         while True:
368             for sequence in color_sequences:
369                 RL.setDifferentColors(sequence)
370                 time.sleep(ANIMATION_DELAY)
371     except KeyboardInterrupt:
372         # Safe shutdown process
373         RL.pause()
374         RL.led_close()
375         print("\nhe lighting animation has safely stopped")
376         sys.exit(0)
377     except Exception as e:
378         print(f"Unknown error occurred: {e}")
379         RL.pause()
380         RL.led_close()
381         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.