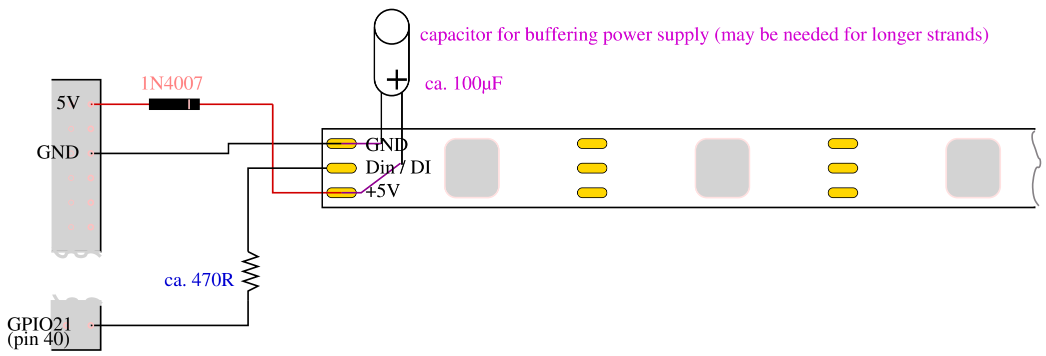
Using a RPi for driving WS2812B LED strands

Mattscheibe, January 2021

This is more expensive than using an arduino clone or an AVR stand alone controller for driving WS2812B LEDs. It is easy though, to port a prototype program, developed on a Raspberry Pi Zero, for example, to a battery powered AVR controller that sits inside the final device.

(This library can also be found for Arduino boards and for standalone AVR controllers.)

Driving LED strands is easy. You need to connect two wires for power supply (+5V and GND) and one wire for signal input (labeled Din on strand). LED strands have current control built in for every pixel. A pixel at the end of a strand will, if it lights up at all, lights up with the same brightness as the first LED on the strand (if both are set to the same brightness value, that is to say). Yet signal level on Din should not diverge too much from power supply level. It might be a good idea to put one diode between Raspberry Pi's 5V output und LED strand's 5V input, thus lowering Pi's 5V by around 0.7V, getting it close to the 3.3V signal level on Pi's outputting GPIO pin. A resistor should drive the Din signal input, otherwise the first LED of the strand might get bricked (has happened to me, might not happen to you). A small electrolytical capacitor might be necessary to buffer power supply to the stripe. A 1N4007 diode can take up to 1A of current. You shouldn't drag more out your poor Pi, anyways. Dim LEDs, for that end. A maximum brightness of 30 or maybe 50 (out of 255) should usually be more that enough.


capacitor for buffering power supply (may be needed for longer strands)
ca. 100µF
1N4007
5V
GND
GND
Din / DI
+5V
ca. 470R
GPIO21
(pin 40)

Interface library is this one (https://github.com/Mattscheibe/ws2812b_rpi), it uses low level functions of another library (https://github.com/jgarff/rpi_ws281x). Both need to be installed to the RPi. My impression is that J. Graff's library is super stable and versatile, yet its C API is not as "accessible" as this library's "wrapper API".
ws2812b_rpi will not support strands of non-WS2812b LEDs, while J. Graff's library does. (I only go for the cheapest. . .)

Usage:

1. `sudo apt-get install cmake ; cd /tmp ; git clone https://github.com/jgarff/rpi_ws281x ; cd rpi_ws281x ; mkdir build ; cd build ; cmake .. ; sudo make install`
2. `cd /tmp ; git clone https://github.com/Mattscheibe/ws2812b_rpi ; cd ws2812b_rpi ; sudo make install`
3. `mkdir ~/Test ; cp /tmp/ws2812b_rpi/Examples/Makefile ~/Test ; cd ~/Test`
4. Type "`make make`" and set a target name and a main source name. (If you prefer to use a different editor, edit this entry, too.)
5. Type "`make edit`" and enter your program. It should start with

   ```
   #include <ws2812b_rpi.h>
   ```
6. Save your program, leave the editor, and type "`make run`" to start it.

   If you are using gedit *you might go to "Manage External Tools " and add a command with an entry "make run" and an appropriate keyboard short cut (e.g. STRG-U) and tell it to save on every execution. Other GUI editors might have similar functions.*

Example:

```
#include <ws2812b_rpi.h>
int main() {
  initLEDsPCM(1,1,1); // very small "matrix": 1 row, 1 column, type 1 (that is: 1 LED :-)
  setRGB(0,0,0x001100); // green at position 0,0 (the only LED present), according to color scheme RRGGBB
  showLEDs(); // now display all (...) LEDs
  sleep(4); // wait for 4 seconds
  endLEDs(); // this should be put at the end of every program (switches LEDs off and releases RPi interfa-
ces)
  return 0;
}
```

There's a table of all available commands at the flipside of this page.

For complete programs with *input* and *output*, *key entry* etc. you might want to include **wiringPi** library (also: cf. flip side of this page).

API:

`initLEDsPCM(`*`width`*`,`*`height`*`,`*`type`*`)` needs to be called first in every program, it establishes a connection between Pi's output and strand's input and it clears all LEDs—for type see below

`setR(`*`row`*`,`*`column`*`,`*`colorValue`*`)` sets the color value of the pixel at position (row,column) to a red value between 0 ("off") und 255 ("full brightness")

`setG(`*`row`*`,`*`column`*`,`*`colorValue`*`)` dito for green

`setB(`*`row`*`,`*`column`*`,`*`colorValue`*`)` dito for blue

`setRGB(`*`row`*`,`*`column`*`,`*`colorValue`*`)` sets the color value of the pixel at position (row,column) to an RGB color value between `0x00000` and `0xFFFFFF`; with positions "`0xRR`*`GG`*`BB`", i.e. first red **RR**, then green *GG*, then blue **BB**. Prefix "`0x`" allows for hexadecimal entry of color values. Alternatively, you might calculate a single decimal value—or use binary values of this shape: "`0b 00000000 11111111 00000000`".

`showLEDs()` displays the current content of the color arry to the pixels—it is best called only after all color values are set or updated to a current value

`clearLEDs()` clears all LEDs ("off") and also resets the color value array (clear lights and clear memory)

`getR(`*`row,column`*`)` gets red value (between 0 and 255) of pixel at position (row,column).

`getG(`*`row,column`*`)` dito for green

`getG(`*`row,column`*`)` dito for blue

`getRGB(`*`row,column`*`)` gets a decimal value (comprising red, green and blue) of pixel at position (row,column)

`setMaxBrightness(`*`maximumColorValue`*`)` sets an upper limit to the maximum brightness. All color values will be scaled so that they fit in the range between 0 and this maximum value. Can be used to dim LEDs (good for Pi, good for the eye). Values range between 0 (not very useful, everything will stay dark) and 255 (no dimming at all). If prototyping with a bad power supply that cannot drive much current, set this to a low value.

`endLEDs`() call this at the end of your programm to switch off all LEDs and to free Pi's interfaces again (especially if run in PWM mode or SPI mode)

Type setting in initLEDs(width,height,type) needs to reflect your wiring scheme:

```
Type  0: °|_|‾↓   1: °|/|/↓   2: ↑_|‾|_|° 3: ↓\|\|°

Type  4: °-       5: °-       6: -°       7: -°
           |           /         |             \
           -           -          -            -
          |            /           |           \
           →           →          ←            ←

Type   8: .|‾|_↑   9: .|\|\↑  10: ↑_|‾|.  11: ↑/|/|.

Type 12: →         13: →       14: ←       15: ←
         |              \            |           /
          -             -          -            -
           |            \          |            /
         .-            .-          -.           -.

° or . = signal entry , | = strand , | = cross connection
```

Using this ws2812b-library in its other variants will help you to run your programs almost unchanged on either, an AVR controller or, an Arduino board. What would be different, is only the `#include` *line, the* Makefile*, of course, and* `initLEDs()` *would be called without parameters for AVR, there would also be no* `endLEDs();` *at the end.*

Arduino boards could drive more than one strand.

When including RPi's wiringPi library (with `#include <wiringPi.h>`*), you get access to many useful GPIO and timing commands, for key input, gereral signal input / output etc. . If you intend to make use of this library, uncomment its corresponding line in* Makefile*.*