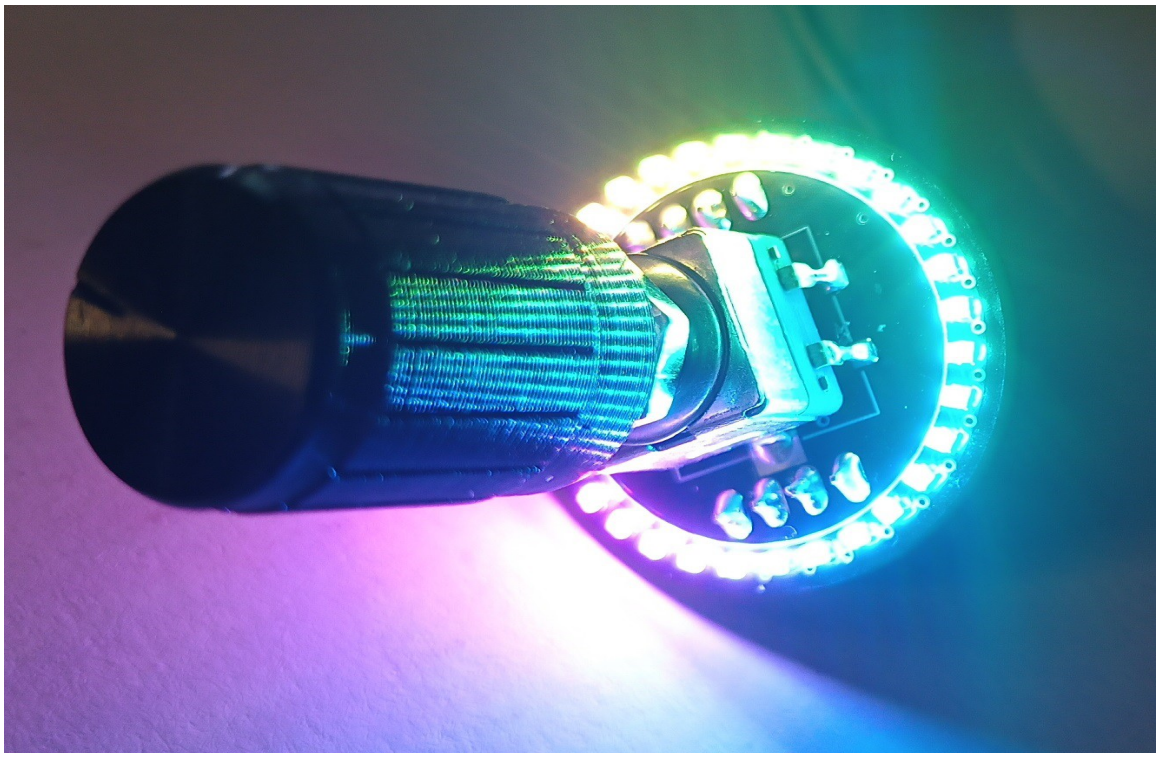
Raging Bits RGB Rotary module v1.1

Top level specs

Serial port interface.
32 addressable LEDs
360º rotary



Device interface

Input Power - 5V Supply.
Serial interface 115200.

Normal Work
UART Header – Communications for control and interface
TX – UART output data
RX – UART input data
5V – 5 volts input power
GND – Ground

Advanced user ONLY
SWD Header - (USE NOT ADVISED) STM32 programming port
CLK – SWD clock line
SDA – SWD data line
3V3 – 3.3Volt input/output for internal LDO
GND – Ground

!!! IMPORTANTE! NEVER POWER 5V and 3.3V bus SIMULTANEOUSLY !!!
This could damage the device!

Commands

Before anything else, the system comes ready to work. No further modifications are needed!
The UART is fixed at 115200, 1 start bit, 1 stop bit, no control flow.

Supported commands by the original application firmware.

Unit to user:

'A' – Acknowledge. This isn't a command, rather a reply to a command or data frame.
Commands must be waited for up to 5 seconds.

')' - Rotary rotated 1 click clockwise.

'(' - Rotary rotated 1 click counter-clockwise.

'p' – Rotary Button pressed.

'P' – Rotary Button released.

User to Unit:
'R' - Reset
This command will make the unit reset imediately. No ack is given to this command.

<R
<{wait at least 0.5s}

'D' – Animation upload binary
This command sets the unit ready to receive a new binary animation file.
This command will have an ack given with must be waited.
Once this command is accepted, there will be a MAXIMUM of 2 seconds between 8 bytes frames.
If the command times out, it needs to be restarted.
ATTENTION, this command will erase any previous animation sent.

<D
>A
<{8bytes}
>A
<{8bytes}
>A
...
<{8bytes}
>A
< {nothing, wait timeout}

'F' – LED data upload request.
This command sets the unit ready to receive a new LED data frame of 32 x GRB data bytes, 96 bytes total.
The timeout for each byte to be received by the unit before the command is finished and the present data set to the LEDs, is of 5ms. This is a demand to keep the framerate.
The unit will automatically send an ack when a timeout happens, and accept the current data in the frame up to date.
When the command times out, a new frame can be started right after the ack - 'A'.

<F
>A
<{96 bytes organized in 3bytes representing GRB data}
> Frame sent to LEDs
>A

'I' – Intensity setting.
This command sets the current intensity of the LEDs and consists on sending the command with one extra byte with the intensity to be est.
The intensity can be set from 0 to 128.

<I
<{1byte with value between 0 and 128}
>A

Auxiliary support

There are support python files that have implemented or have working examples on how to interface the unit.
The python files use Python 3 and PySerial library to work, make sure to install it along with python.

Animation upload to the unit

There are a available animations that can be uploaded to the unit so they can be executed by the unit in standalone operation.
To do this, follow the 'D' – Animation upload binary proceedure.
The python file that already implements this as an example, is 'animation_load.py'.
To invoke the uploader, the both the COM port and the animation binary file must be indicated.

Windows example:
python3 animation_load.py COM4 gradient1.bin

Linux ecample:
python3 animation_load.py /dev/ttyUSB1 gradient1.bin

NOTE
DO NOT use the rotary or any other commands during this process. If will likely fail the upload.
Once the upload is finished, the unit will bring up the animation in the next 5 seconds.

Animation streaming to the unit

There are a available examples on how to use the animation streaming command to send data to the unit, performing the animations from the user PC/PI/Other, in file
python_animation_stream_examples.py

At the top of the file is located the serial port selection. Do not change the speed.
At the bottom of the file, is the invoking of the respective animation. Change the invoke call to the wanted animation in the example.

Example:
Running trail indicator:

```
if __name__ == "__main__":
   try:
      # Change this to any animation
       run_animation(trail_indicator_frame, intensity=1.0) <---------- Call the trail animation
   except KeyboardInterrupt:
       print("\nExiting.")
```

Running fire flicker:

```
if __name__ == "__main__":
    try:
        # Change this to any animation
        run_animation(fire_flicker_frame, intensity=1.0) <---------- Call the fire flicker animation
    except KeyboardInterrupt:
        print("\nExiting.")
```

Etc...

Hardware information

!!!ATTENTION!!!

This chapter is for ADVANCED USERS only!!!
The original firmware IS NOT AVAILABLE nor is allowed to be used by anyone that not in Raging Bits producs.

IF the user erases/reprograames over the device application firmware, in order to reinstate this original device application firmware, the device will need to be sent to Raging Bits in order to be reprogrammed.

This chapter exposes the internal microcontroller connections and system setup.

This chapter is dedicated to those whom may want to use the unit as a development platform.

Every device will have the main application code protected from readback. To use the unit as development platform, the unit needs to be unlocked through the options bytes that WILL ERASE the main application from the internal memory.

Microcontroller – STM32G030C8T6 – NO EXTERNAL Crystals, use internal RC oscilators.

LEDs Data IN - STM32 pin16 – PA11 through 120ohm resistor.

Serial UART
UART pins have 120ohm resistors.
UART RX – STM32 pin 1 – PB0/PB1/PB2/PA8 – UART 1
UART TX – STM32 pin 20 – PB3/PB4/PB5/PB6/PB7/PB8 – UART 1

Encoder
Encoder pin pulls to GND when active. No external pull ups exist, use internal STM pull ups.
Data out 1 – STM32 pin 7 – PA0
Data out 2 – STM32 pin 8 – PA1
Button – STM32 pin 9 – PA2

Debugger/Programmer mode – SWD (NOT JTAG)
SWD IO - STM32 pin18 – PA13
SWD CLK - STM32 pin 19 – PA14/PA15

References

STM32G030Cx datasheet.
STM32G030Cx manual.
WS2812B datasheet.