Point 1: Configuration of Number of LED Drivers.

The GMMK PRO has 2 LED drivers on board, while other keyboards can have 1, 2, or even more. Therefore, the SW drivers must be scalable to any number of HW LED drivers.

AW20216	AW20216S (Glorious)
Configuration of new LED drivers is done in the file **aw20216.c** The SW driver does not have the same schema to add new LED drivers as the existing LED drivers (e.g., IS31FL3731).	Configuration of new LED drivers is done in the file **rgb_matrix_drivers.c** The SW driver follows the same schema as the existing LED drivers (e.g., IS31FL3731).

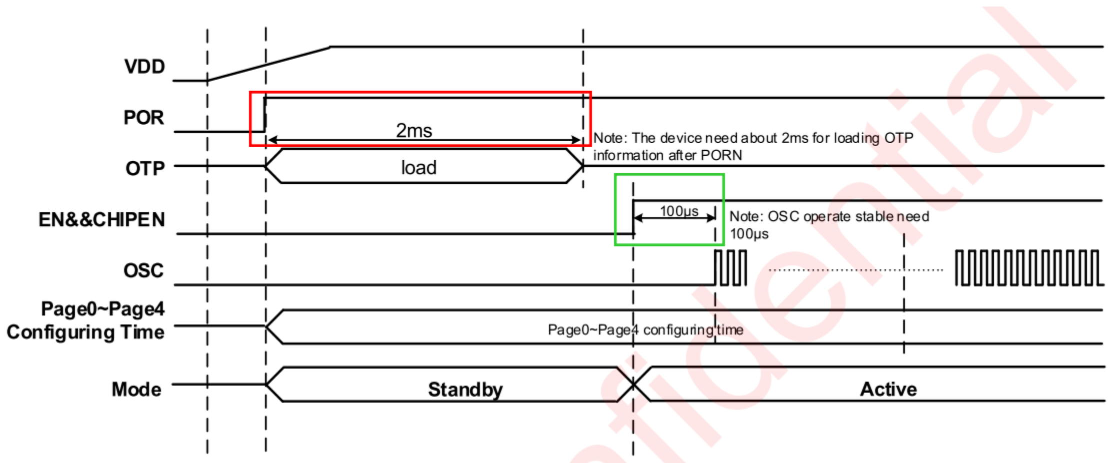
Point 2: Send Brightness (PWM Value) to the LEDs.

The MCU periodically sends the value of the brightness to every LED, these transfers must be done as fast as possible.

AW20216	AW20216S (Glorious)
The function **AW20216_update_pwm** sends so many SPI transfers as LED values to update. There is an overhead of 2 bytes per byte sent.	The function **AW20216S_write_pwm_buffer** sends only one SPI transfer per driver. The overhead is minimum (2 bytes per 256 bytes sent).

Point 3: Initialization Timing

The MCU must wait a minimum amount of time before communicating with the HW LED driver. This is clearly stated in the datasheet:



VDD
POR
2ms
Note: The device need about 2ms for loading OTP information after PORN
OTP
load
EN&&CHIPEN
100μs
Note: OSC operate stable need 100μs
OSC
Page0~Page4 Configuring Time
Page0~Page4 configuring time
Mode
Standby
Active

Figure 9 Power up Timing

The sequence to be implemented is:

- Wait for at least 2ms. (POR)
- Set enable pin to high level.
- Wait for at least 100us. (ENABLE)
- The LED driver is ready.

AW20216	AW20216S (Glorious)
No implementation of start-up timing.	The function **AW20216S_enable** implements these delays: - For the POR delay, it waits for 5ms. - For the EN delay, it waits for 500us

Point 4: CS Set High Before Any Communication.

It is usual to have multiple SPI slaves connected to the same SPI instance in the MCU (as for gmmk/pro keyboard), that means that all those SPI slaves share the signals MOSI, MISO and SCK.

Before any communication to any SPI slave connected to the same SPI instance of the MCU, all CS signals must be de-asserted (in this case set to high level).

AW20216	AW20216S (Glorious)
The function **AW20216_init** initializes every HW LED driver one after another, de-asserting the corresponding CS signal before the transfer, but it	The function **init** in the file **rgb_matrix_drivers.c** split the init process into two parts, the first part enables the SPI slaves and de-assert the CS signals

does not care about the CS state of the other SPI slaves.	and the second part transfers the configuration to the SPI slaves in a safe way.

Point 5: Delay Between Register Writes

The SPI transfers must be separated for a minimum amount of time. This amount of time is normally specified in the datasheet of the SPI slave, but this LED driver does not specify it at all.

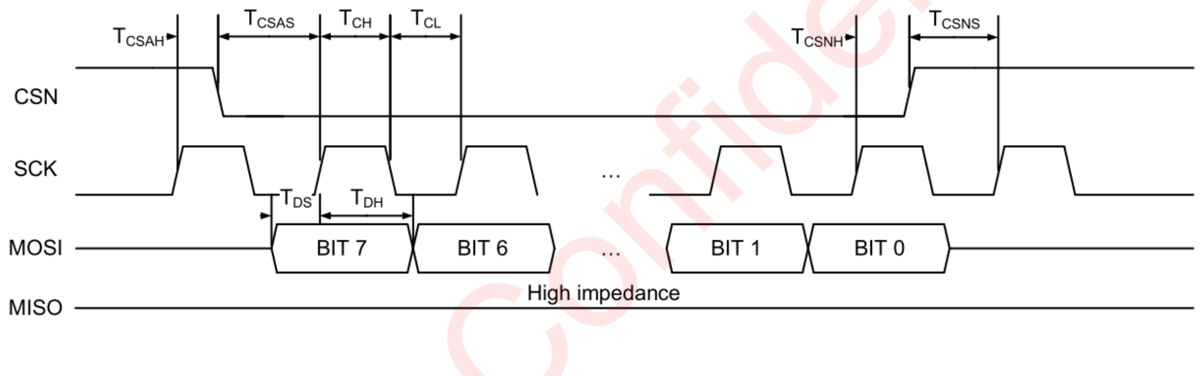


Figure 7 SPI Interface Input Timing

It does not mean that this time is 0, simply that is not specified in the datasheet.

A similar time parameter is present in the datasheet, labelled as Tcsns, that is the minimum time to disregard a clock edge outside the SPI transfer.

AW20216	AW20216S (Glorious)
It assumes that the delay is 0.	As the delay between register writes is only used in the init phase, this delay does not affect the performance of the driver. As no info is present in the datasheet, a value of 100us is used.

Point 6: Configuration of the Number of LED Lines Controlled by the LED Driver

The register GCR has a bitfield named SWSEL to select how many lines of LED are active. The gmmk/pro has 12 active lines for the first LED driver and 11 active lines for the second LED driver. Another keyboard can have a different number of active lines per driver, and as you can see can be different between LED drivers.

AW20216	AW20216S (Glorious)
Uses a constant value for the register GCR inside the file **AW20216.c** that is applied to all drivers, not allowing the asymmetric configuration of the gmmk/pro keyboard.	In the file **config.h** (part of the gmmk/pro keyboard configuration) there are 2 constant to define the active lines: - SW_LINES_ENABLE_DRIVER_1 - SW_LINES_ENABLE_DRIVER_2 This allows maximum flexibility to the driver.