Brushless setup - driving esc's - firmware settings. Revised Feb, 2018.

This page follows on from the first brushless page with a little more detail about setting up a silverwared FC to control a quad with brushless motors.

There are 2 basic ways to drive esc's. It is highly recommended to use BLHeli_S esc's to make things easy and effective. The reason is that they can use Dshot or the older PPM protocols. Either protocol works fine. For Dshot, the esc drive signal must come from the FET gate and you would select dshot_driver_beta in hardware.h. If you do not plan to use the signal from the FET gate (i.e. you will use pull-up resistors after the FET's) then you must use PPM protocol (e.g. oneshot) and select esc_driver in hardware.h. This article refers only to BLHeli_S esc's. For other esc's, please ask questions in the H8 mini thread if you need something explained.

If you are familiar with Betaflight or Cleanflight, you might wonder what “PPM” is. In silverware, a PPM signal is a conventional signal that drives an esc using (for example) oneshot protocol, and a PWM signal is one that drives an FET for a brushed motor.

If you have BLHeli flashable esc's and want to connect to it to check or change the settings, you will need an interface between BLHeliSuite and the esc. Many people use an Arduino Nano, which is only a few dollars, and the Nano can be flashed by BLHeliSuite to become the interface to connect to many different esc's. An alternative is to buy an esc programmer for your particular esc's, and this will serve as the BLHeliSuite/esc interface instead of the Arduino Nano.

How to set up the Dshot esc driver:

The Dshot esc driver is selected in hardware.h. Then, in drv_dshot.c, select either DSHOT600, DSHOT300 or DSHOT150, and adjust the IDLE_OFFSET number if you need to, in order to set the desired “idle” speed. Default is 40 and I often use 32 for high-Kv 100mm and 120mm quads.

Note for Dshot: The signal can only be taken from before the FET (FET gate) for it to work.

How to set up the PPM esc driver:

In **hardware.h** - the esc PPM driver is selected instead of the default PWM driver

Change the code in the **hardware.h** file to look like this to enable the esc PPM driver:

```
//#define USE_PWM_DRIVER
#define USE_ESC_DRIVER
//#define USE_DSHOT_DRIVER
```

In **drv_esc.c** - the esc min throttle, esc max throttle and throttle_off values are set

The default values for BLHeli_S are Min Throttle 1148 and Max Throttle 1832.

The values in drv_esc.c need to be set so that the FC values match what the esc is expecting, so ESC_MAX can be set to 1832 and ESC_MIN is set a bit higher than what the esc is expecting, to raise the idle speed a little, and usually a value of 1152 to 1160 works well. It can be changed/tuned to suit your particular quad and preferences.

```
#define ESC_MIN 1160
#define ESC_MAX 1832
```

ESC_THROTTLEOFF is the value sent to the esc by the FC to to indicate the motors off condition. Anything under 1148 should work but I often use 960 to be sure.

```
#define ESC_THROTTLEOFF 960
```

Finally, the PPM signal polarity needs to be set in drv_esc.c. If you are taking the signal from after the FET, from the motor output - (neg) pad, the signal must be inverted in the firmware because the FET also inverts it, (by inverting twice, the signal ends up having the correct polarity).

Setting the esc signal polarity for H8 mini blue and CG023:

If you are taking the signal from after the FET with pullup resistors to drive the esc, the polarity would be set inverted:

```
// invert = signal after fets (may need 1k pullup resistor)
// commented = signal straight from CPU pins
#define ESC_INVERT_SIGNAL
```

If you are taking the signal from the gate of the FET (straight from the CPU pins) to drive the esc, the polarity would be set non-inverted, like this:

```
// invert = signal after fets (may need 1k pullup resistor)
// commented = signal straight from CPU pins
//#define ESC_INVERT_SIGNAL
```

Setting the esc signal polarity for H8 mini green and H101:

If you are taking the signal from after the FET with pullup resistors to drive the esc, the polarity would be set inverted:

```
// output polarity ( low - motor output with pullup resistor (500 ohms or
near) )
// enable for motor output after fets
#define INVERTED_PWM
```

If you are taking the signal from the gate of the FET (straight from the CPU pins) to drive the esc, the polarity would be set non-inverted, like this:

```
// output polarity ( low - motor output with pullup resistor (500 ohms or
near) )
// enable for motor output after fets
//#define INVERTED_PWM
```

PWM

For PWM (brushed motor ouput), a BLHeli flashable esc has to be used to enable the pwm input of the esc. To flash an esc, or just connect to it to check or change the settings, you will need an interface between BLHeliSuite and the esc. Many people use an Arduino Nano, which is only a few dollars, and it can be flashed by BLHeliSuite to become the interface to connect to many different esc's. An alternative is to buy an esc programmer for your particular esc's, and this will serve as the BLHeli/esc interface instead of the Arduino Nano.

When you connect an esc to BLHeliSuite, there is a box labelled “Enable PWM Input”. In this box, the slider must be set to ON and the setting written to the esc by clicking on the Write Setup button. Then confirm the settings have been written by clicking on the Read Setup button. Note: BLHeli_S esc's do not support PWM Input. For BLHeli_S you must use the PPM method to drive the esc.

If you are using resistors (and therefore taking the signal for the esc's after the FET), the Input Polarity in

BLHeliSuite must be set to Negative. If you are taking the signal from the gate of the FET, Input Polarity would be set to the default setting Positive.

PWM FIRMWARE SETTINGS

For PWM there are only a couple of things to change in the firmware in **config.h**. The motor PWM has to be set to 1kHz, 2kHz, 4kHz, 8kHz or 12kHz. These are the values that BLHeli firmware will accept. I have used mostly 8kHz. With PWM, you have to manually arm the esc by raising and then lowering the throttle.

```
For the H8 mini green and H101 board set (in config.h):
#define MOTOR_CURVE_NONE
#define PWM_8KHZ

For H8 mini blue board and CG023 set (in config.h):
#define PWMFREQ 8000
#define MOTOR_CURVE_NONE
```

For the H8 mini green, H8 mini blue, H101 and CG023 code, the min motor speed is also changed in config.h. The line MOTOR_MIN_ENABLE has to be uncommented, and the MOTOR_MIN_VALUE has to be set, usually to a value between 0.06 and 0.12 (lower for 3S and higher for 2S battery). This sets the minimum motor speed that the motors will spin at, and if it is too low, the motors will not start together cleanly. If it is too high, the quad may lift off at minimum motor speed, so start with a low value and work up. Silverxxx recommends testing it with the “motors_to_throttle” option, as that way you can clearly see if they start and run properly.

```
// limit minimum motor output to a value (0.0 - 1.0)
#define MOTOR_MIN_ENABLE
#define MOTOR_MIN_VALUE 0.07
```