Led Strip

Cleanflight supports the use of addressable LED strips. Addressable LED strips allow each LED in the strip to be programmed with a unique and independant color. This is far more advanced than the normal RGB strips which require that all the LEDs in the strip show the same color.

Addressable LED strips can be used to show information from the flight controller system, the current implementation supports the following:

- Up to 32 LEDs.
- Indicators showing pitch/roll stick positions.
- Heading/Orientation lights.
- Flight mode specific color schemes.
- Low battery warning.

The function and orientation configuration is fixed for now but later it should be able to be set via the UI or CLI..

In the future, if someone codes it, they could be used to show GPS navigation status, thrust levels, RSSI, etc. Lots of scope for ideas and improvements.

Likewise, support for more than 32 LEDs is possible, it just requires additional development.

Supported hardware

Only strips of 32 WS2812 LEDs are supported currently. If the strip is longer than 32 leds it does not matter, but only the first 32 are used.

WS2812 LEDs require an 800khz signal and precise timings and thus requires the use of a dedicated hardware timer.

Note: The initial code may work with WS2801 + External LEDs since the protocol is the same, WS2811/WS2812B should also work but may require very simple timing adjustments to be made in the source. Not all WS2812 ICs use the same timings, some batches use different timings.

It could be possible to be able to specify the timings required via CLI if users request it.

Connections

WS2812 LED strips generally require a single data line, 5V and GND.

WS2812 LEDs on full brightness can consume quite a bit of current. It is recommended to verify the current draw and ensure your supply can cope with the load. On a multirotor that uses multiple BEC ESC's you can try use a different BEC to the one the FC uses. e.g. ESC1/BEC1 -> FC, ESC2/BEC2 -> LED strip. It's also possible to power one half of the strip from one BEC and the other half from another BEC. Just ensure that the GROUND is the same for all BEC outputs and LEDs.

Target	Pin	Led Strip	Signal
Naze/Olimexino	RC5	Data In	PA6
CC3D	???	Data In	PB4
ChebuzzF3/F3Discovery	PB8	Data In	PB8

Since RC5 is also used for SoftSerial on the Naze/Olimexino it means that you cannot use softserial and led strips at the same time. Additionally, since RC5 is also used for Parallel PWM RC input on both the Naze, Chebuzz and STM32F3Discovery targets, led strips can not be used at the same time at Parallel PWM.

Configuration

Enable the `LED_STRIP` feature via the cli:

```
feature LED_STRIP
```

If you enable LED_STRIP feature and the feature is turned off again after a reboot then check your config does not conflict with other features, as above.

Configure the LEDs using the `led` command.

The `led` command takes either zero or two arguments - an zero-based led number and a pair of coordinates, direction flags and mode flags.

If used with zero arguments it prints out the led configuration which can be copied for future reference.

Each led is configured using the following template: `x,y:ddd:mmm`

`x` and `y` are grid coordinates of a 0 based 16x16 grid, north west is 0,0, south east is 15,15 `ddd` specifies the directions, since an led can face in any direction it can have multiple directions. Directions are:

`N` - North `E` - East `S` - South `W` - West `U` - Up `D` - Down

For instance, an LED that faces South-east at a 45 degree downwards angle could be configured as `SED`.

Note: It is perfectly possible to configure an LED to have all directions `NESWUD` but probably doesn't make sense.

`mmm` specifies the modes that should be applied an LED. Modes are:

- `W` - Wwarnings.
- `F` - Flight mode & Orientation
- `I` - Indicator.
- `A` - Armed state.
- `T` - Thrust state.

Example:

```
led 0 0,15:SD:IAW
led 1 15,0:ND:IAW
led 2 0,0:ND:IAW
led 3 0,15:SD:IAW
```

to erase an led, and to mark the end of the chain, use `0,0::` as the second argument, like this:

```
led 4 0,0::
```

Modes

Warning

This mode simply uses the leds to flash when warnings occur.

- Battery warning flashes the LEDs between red and off when the battery is low if battery monitoring is enabled.
- Failsafe warning flashes the LEDs between light blue and lime green when failsafe is active.

Flight Mode & Orientation

This mode shows the flight mode and orientation.

When flight modes are active then the leds are updated to show different colors depending on the mode, placement on the grid and direction.

Leds are set in a specific order: * Leds that marked as facing up or down. * Leds that marked as facing west or east AND are on the west or east side of the grid. * Leds that marked as facing north or south AND are on the north or south side of the grid.

That is, south facing leds have priority.

Indicator

This mode flashes LEDs that correspond to roll and pitch stick positions. i.e. they indicate the direction the craft is going to turn.

Armed state

This mode toggles LEDs between green and blue when disarmed and armed, respectively.

Note: Armed State cannot be used with Flight Mode.

Thrust state

This mode fades the LED current LED color to the previous/next color in the HSB color space depending on throttle stick position. When the throttle is in the middle position the color is unaffected, thus it can be mixed with orientation colors to indicate orientation and throttle at the same time.

Positioning

Cut the strip into sections as per diagrams below. When the strips are cut ensure you reconnect each output to each input with cable where the break is made. e.g. connect 5V out to 5V in, GND to GND and Data Out to Data In.

Orientation is when viewed with the front of the aircraft facing away from you and viewed from above.

Example 12 LED config

The default configuration is as follows

```
led 0 2,2:ES:IA
led 1 2,1:E:WF
led 2 2,0:NE:IA
led 3 1,0:N:F
led 4 0,0:NW:IA
led 5 0,1:W:WF
led 6 0,2:SW:IA
led 7 1,2:S:WF
led 8 1,1:U:WF
led 9 1,1:U:WF
led 10 1,1:D:WF
led 11 1,1:D:WF
```

Which translates into the following positions:

```
     5             3
      \           /
       \    4    /
        \ FRONT /
      6 | 9-12  | 2
        /  BACK \
       /    8    \
      /           \
     7             1
```

LEDs 1,3,5 and 7 should be placed underneath the quad, facing downwards. LEDs 2, 4, 6 and 8 should be positioned so the face east/north/west/south, respectively. LEDs 9-10 should be placed facing down, in the middle LEDs 11-12 should be placed facing up, in the middle

This is the default so that if you don't want to place LEDs top and bottom in the middle just connect the first 8 leds.

Example 16 LED config

```
15,15:SD:IA
8,8:E:FW
8,7:E:FW
15,0:ND:IA
7,7:N:FW
8,7:N:FW
0,0:ND:IA
7,7:W:FW
7,8:W:FW
0,15:SD:IA
7,8:S:FW
8,8:S:FW
7,7:D:FW
8,7:D:FW
7,7:U:FW
8,7:U:FW
```

Which translates into the following positions:

```
      7             4
       \           /
        \   6-5   /
       8 \ FRONT / 3
          | 13-16 |
       9 /  BACK \ 2
        /  11-12  \
       /           \
     10             1
```

LEDs 1,4,7 and 10 should be placed underneath the quad, facing downwards. LEDs 2-3, 6-5, 8-9 and 11-12 should be positioned so the face east/north/west/south, respectively. LEDs 13-14 should be placed facing down, in the middle LEDs 15-16 should be placed facing up, in the middle

Exmple 28 LED config

```
#right rear cluster
led 0 9,9:S:FWT
led 1 10,10:S:FWT
led 2 11,11:S:IA
led 3 11,11:E:IA
led 4 10,10:E:AT
led 5 9,9:E:AT
# right front cluster
led 6 10,5:S:F
led 7 11,4:S:F
led 8 12,3:S:IA
led 9 12,2:N:IA
led 10 11,1:N:F
led 11 10,0:N:F
# center front cluster
led 12 7,0:N:FW
led 13 6,0:N:FW
led 14 5,0:N:FW
led 15 4,0:N:FW
# left front cluster
led 16 2,0:N:F
led 17 1,1:N:F
led 18 0,2:N:IA
led 19 0,3:W:IA
led 20 1,4:S:F
led 21 2,5:S:F
# left rear cluster
led 22 2,9:W:AT
led 23 1,10:W:AT
led 24 0,11:W:IA
led 25 0,11:S:IA
led 26 1,10:S:FWT
led 27 2,9:S:FWT
```

```
       17-19  10-12
20-22 \           / 7-9
       \  13-16  /
        \ FRONT /
        /  BACK \
       /         \
23-25 /           \ 4-6
       26-28   1-3
```

All LEDs should face outwards from the chassis in this configuration.

Note: This configuration is specifically designed for the Alien Spider AQ50D PRO 250mm frame.

http://www.goodluckbuy.com/alien-spider-aq50d-pro-250mm-mini-quadcopter-carbon-fiber-micro-multicopter-frame.html

Troubleshooting

On initial power up the LEDs on the strip will be set to WHITE. This means you can attach a current meter to verify the current draw if your measurement equipment is fast enough. This also means that you can make sure that each R,G and B LED in each LED module on the strip is also functioning.

After a short delay the LEDs will show the unarmed color sequence and or low-battery warning sequence.

If the LEDs flash intermittently or do not show the correct colors verify all connections and check the specifications of the LEDs you have against the supported timings (for now, you'll have to look in the source).

Also check that the feature `LED_STRIP` was correctly enabled and that it does not conflict with other features, as above.

Board - Naze32

The Naze32 target supports all Naze hardware revisions. Revison 4 and Revision 5 are used and frequently flown by the primary maintainer. Previous Naze hardware revisions may have issues, if found please report via the github issue tracker.

Serial Ports

Value	Identifier	Board Markings	Notes
1	USART1	RX/TX / TELEM	Also connected to USB port via CP2102 IC
2	USART2	RC3/4	
3	SoftSerial 1	RC5/6	
4	SoftSerial 2	RC7/8	

Board - CC3D

The OpenPilot Copter Control 3D aka CC3D is a board more tuned to Acrobatic flying or GPS based auto-piloting. It only has one sensor, the MPU6000 SPI based Accelerometer/Gyro. It also features a 16mbit SPI based EEPROM chip. It has 6 ports labelled as inputs (one pin each) and 6 ports labelled as motor/servo outputs (3 pins each).

If issues are found with this board please report via the github issue tracker.

The board has a USB port directly connected to the processor. Other boards like the Naze and Flip32 have an on-board USB to uart adapter which connect to the processor's serial port instead.

Currently there is no support for virtual com port functionality on the CC3D which means that cleanflight does not use the USB socket at all.

Serial Ports

Value	Identifier	Board Markings	Notes
1	USART1	MAIN PORT	Has a hardware inverter for SBUS
2	USART3	FLEX PORT	

Software serial is not supported yet due to timer and pin configuration mappings.

To connect the GUI to the flight controller you need additional hardware attached to the USART1 serial port (by default).

Flashing

There are two primary ways to get Cleanflight onto a CC3D board.

- Single binary image mode - best mode if you don't want to use OpenPilot.
- OpenPilot Bootloader compatible image mode - best mode if you want to switch between OpenPilot and Cleanflight.

Single binary image mode.

The entire flash ram on the target processor is flashed with a single image.

OpenPilot Bootloader compatible image mode.

The initial section of flash ram on the target process is flashed with a bootloader which can then run the code in the remaining area of flash ram.

The OpenPilot bootloader code also allows the remaining section of flash to be reconfigured and re-flashed by the OpenPilot Ground Station (GCS) via USB without requiring a USB to uart adapter.

In this mode a USB to uart adapter is still required to connect to via the GUI or CLI.# Sonar

Asonar sensor can be used to measure altitude for use with altitude hold modes.

The current sensor takes over from the pressure sensor at low altitudes, but only when the angle of the aircraft is small.

Hardware

Currently the only supported sensor is the HCSR04 sensor.

Connections

Naze/Flip32+

Mode	Trigger	Echo	Inline 1k resistors
Parallel PWM	PB8 / Motor 5	PB9 / Motor 6	NO (5v tolerant)
PPM/Serial RX	PB0 / RC7	PB1 / RC8	YES (3.3v input)

Current meter cannot be used in conjunction with Parallel PWM and Sonar.

Olimexino

Trigger	Echo	Inline 1k resistors
PB0 / RC7	PB1 / RC8	YES (3.3v input)

Current meter cannot be used in conjunction sonar.

Receivers (RX)

Parallel PWM

8 channel support, 1 channel per input pin. On some platforms using parallel input will disable the use of serial ports and softserial making it hard to use telemetry or gps features.

PPM (PPM SUM)

12 channels via a single input pin, not as accurate or jitter free as methods that use serial communications.

MultiWii serial protocol (MSP)

Allows you to use MSP commands as the RC input. Only 8 channel support to maintain compatibility with MSP.

Spektrum

12 channels via serial currently supported.

SBUS

12 channels via serial currently supported.

SUMD

16 channels via serial currently supported.

SUMH

8 channels via serial currently supported.

Configuration

See the Configuration document some some RX configuration examples.

For Serial RX enable `RX_SERIAL` and set the `serialrx_provider` cli setting as follows.

Serial RX Provider	Value
SPEKTRUM1024	0
SPEKTRUM2048	1
SBUS	2
SUMD	3
SUMH	4

PPM/PWM input filtering.

Hardware input filtering can be enabled if you are experiencing interference on the signal sent via your PWM/PPM RX.

Use the `input_filtering_mode` cli setting to select a mode.

Value	Meaning
0	Disabled
1	Enabled

Failsafe

There are two types of failsafe:

1. receiver based failsafe
2. flight controller based failsafe

Receiver based failsafe is where you, from your transmitter and receiver, configure channels to output desired signals if your receiver detects signal loss. The idea is that you set throttle and other controls so the aircraft descends in a controlled manner.

Flight controller based failsafe is where the flight controller attempts to detect signal loss from your receiver.

It is possible to use both types at the same time and may be desirable. Flight controller failsafe can even help if your receiver signal wires come loose, get damaged or your receiver malfunctions in a way the receiver itself cannot detect.

Flight controller failsafe system

The failsafe system is not activated until 5 seconds after the flight controller boots up. This is to prevent failsafe from activating in the case of TX/RX gear with long bind procedures before they send out valid data.

After the failsafe has been forced a landing, the flight controller cannot be armed and has to be reset.

The failsafe system attempts to detect when your receiver looses signal. It then attempts to prevent your aircraft from flying away uncontrollably.

The failsafe is activated when:

a) no valid channel data from the RX via Serial RX. b) the first 4 Parallel PWM/PPM channels do not have valid signals.

There are a few settings for it, as below.

Failsafe delays are configured in 0.1 second steps.

1 step = 0.1sec 1 second = 10 steps

`failsafe_delay`

Guard time for failsafe activation after signal lost.

`failsafe_off_delay`

Delay after failsafe activates before motors finally turn off. If you fly high you may need more time.

`failsafe_throttle`

Throttle level used for landing. Specify a value that causes the aircraft to descend at about 1M/sec.

Use standard RX usec values. See Rx documentation.

`failsafe_min_usec`

The shortest PWM/PPM pulse considered valid.

Only valid when using Parallel PWM or PPM receivers.

`failsafe_max_usec`

The longest PWM/PPM pulse considered valid.

Only valid when using Parallel PWM or PPM receivers.

This setting helps catch when your RX stops sending any data when the RX looses signal.

With a Graupner GR-24 configured for PWM output with failsafe on channels 1-4 set to OFF in the receiver settings then this setting, at it's default value, will allow failsafe to be activated.

Configuration

Cleanflight is configured primarilty using the Cleanflight Configurator GUI.

Both the command line interface and gui are accessible by the connecting to a serial port on the target, be it a USB virtual serial port, physical hardware UART port or a softserial port.

The GUI cannot currently configure all aspects of the system, the CLI must be used to enable or configure some features and settings.

See the Serial section for more information. See the Board specific sections for details of the serial ports available on the board you are using.

GUI

The GUI tool is the preferred way of configuration. The GUI tool also includes a terminal which can be used to interact with the CLI.

https://chrome.google.com/webstore/detail/cleanflight-configurator/enacoimjcgeinfnnnpajinjgmkahmfgb

CLI

Cleanflight can also be configured by a command line interface.

The CLI can be accessed via the GUI tool or by sending a single '#' character to the main serial port.

To exit the CLI without saving power off the flight controller.

To see a list of commands type in 'help' and press return.

To dump your configuration (including the current profile), use the 'dump' command.

See the other documentation sections for details of the cli commands and settings that are available.

Building in windows

1- Install cygwin, and GNU Tools ARM Embedded (make sure make is installed). cygwin: https://cygwin.com/install.html gnu tools arm embedded: https://launchpad.net/gcc-arm-embedded

2- In a terminal go to cleanflight directory

3- make clean

4- rm -rf obj

5- make

This should work.Something like this output should be the result: C:\Users\nico\Documents\GitHub\cleanflight>make %%
startup*stm32f10x* md*gcc.s %% accgyro*adxl345.c %% accgyro*bma280.c %% accgyro*l3g4200d.c %% accgyro*mma845x.c*
*%% accgyro*mpu3050.c %% accgyro*mpu6050.c %% adc*common.c %% adc*stm32f10x.c %% barometer* bmp085.c %%
barometer*ms5611.c %% bus*spi.c %% bus*i2c*stm32f10x.c %% compass*hmc5883l.c %% gpio*stm32f10x.c %% light*ledring.c*
*%% sonar*hcsr04.c %% pwm*mapping.c %% pwm* output.c %% pwm*rssi.c %% pwm* rx.c %% serial*softserial.c %%*
*serial*uart*common.c %% serial* uart*stm32f10x.c %% timer* common.c %% build*config.c %% battery.c %% boardalignment.c*
%% beeper.c %% config.c %% maths.c %% printf.c %% typeconversion.c %% failsafe.c %% main.c %% mw.c %%
*sensors*acceleration.c %% sensors*barometer.c %% sensors*compass.c %% sensors*gyro.c %% sensors*initialisation.c %%
sensors*sonar.c %% bus*i2c*soft.c %% serial* common.c %% sound*beeper.c %% system* common.c %% flight*common.c %%*
*flight*imu.c %% flight*mixer.c %% gps*common.c %% runtime*config.c %% rc*controls.c %% rc*curves.c %% rx* common.c %%
rx*msp.c %% rx* pwm.c %% rx*sbus.c %% rx* sumd.c %% rx*spektrum.c %% telemetry* common.c %% telemetry*frsky.c %%*
telemetry hott.c %% serial*common.c %% serial* cli.c %% serial*msp.c %% statusindicator.c %% core* cm3.c %%
system*stm32f10x.c %% stm32f10x* gpio.c %% stm32f10x*fsmc.c %% stm32f10x* exti.c %% stm32f10x*bkp.c %%*
stm32f10x spi.c %% stm32f10x*adc.c %% stm32f10x* iwdg.c %% misc.c %% stm32f10x*cec.c %% stm32f10x* wwdg.c %%
stm32f10x*tim.c %% stm32f10x* usart.c %% stm32f10x*crc.c %% stm32f10x* flash.c %% stm32f10x*rcc.c %% stm32f10x* sdio.c
%% stm32f10x*i2c.c %% stm32f10x* pwr.c %% stm32f10x*rtc.c %% stm32f10x* can.c %% stm32f10x*dac.c %%*
stm32f10x dbgmcu.c %% stm32f10x*dma.c arm-none-eabi-gcc -o obj/cleanflight*NAZE.elf obj/NAZE/startup*stm32f10x* md*gcc.*
*o obj/NAZE/drivers/accgyro*adxl345.o obj/NAZE/drivers/accgyro*bma280.o obj/NAZE/ drivers/accgyro* l3g4200d.o
obj/NAZE/drivers/accgyro*mma845x.o obj/NAZE/drivers/a ccgyro*mpu3050.o obj/NAZE/drivers/accgyro*mpu6050.o*
*obj/NAZE/drivers/adc*common. o obj/NAZE/drivers/adc*stm32f10x.o obj/NAZE/drivers/barometer*bmp085.o obj/NAZE/
drivers/barometer*ms5611.o obj/NAZE/drivers/bus*spi.o obj/NAZE/drivers/bus*i2c*s tm32f10x.o
obj/NAZE/drivers/compass*hmc5883l.o obj/NAZE/drivers/gpio*stm32f10x.o obj/NAZE/drivers/light*ledring.o*
*obj/NAZE/drivers/sonar*hcsr04.o obj/NAZE/drive rs/pwm*mapping.o obj/NAZE/drivers/pwm* output.o
obj/NAZE/drivers/pwm*rssi.o obj/N AZE/drivers/pwm* rx.o obj/NAZE/drivers/serial*softserial.o obj/NAZE/drivers/seria*
*l*uart*common.o obj/NAZE/drivers/serial* uart*stm32f10x.o obj/NAZE/drivers/timer* common.o obj/NAZE/build*config.o*
obj/NAZE/battery.o obj/NAZE/boardalignment.o ob j/NAZE/beeper.o obj/NAZE/config.o obj/NAZE/common/maths.o
obj/NAZE/common/printf .o obj/NAZE/common/typeconversion.o obj/NAZE/failsafe.o obj/NAZE/main.o obj/NAZE /mw.o
*obj/NAZE/sensors*acceleration.o obj/NAZE/sensors*barometer.o obj/NAZE/sens ors*compass.o obj/NAZE/sensors*gyro.o*
*obj/NAZE/sensors*initialisation.o obj/NAZE /sensors*sonar.o obj/NAZE/drivers/bus*i2c*soft.o obj/NAZE/drivers/serial*common. o
obj/NAZE/drivers/sound*beeper.o obj/NAZE/drivers/system* common.o obj/NAZE/flig ht*common.o obj/NAZE/flight*imu.o
obj/NAZE/flight*mixer.o obj/NAZE/gps*common.o obj/NAZE/runtime*config.o obj/NAZE/rc*controls.o obj/NAZE/rc*curves.o*
obj/NAZE/r x common.o obj/NAZE/rx*msp.o obj/NAZE/rx* pwm.o obj/NAZE/rx*sbus.o obj/NAZE/rx* su md.o
obj/NAZE/rx*spektrum.o obj/NAZE/telemetry* common.o obj/NAZE/telemetry*frsky .o obj/NAZE/telemetry* hott.o
obj/NAZE/serial*common.o obj/NAZE/serial* cli.o obj/ NAZE/serial*msp.o obj/NAZE/statusindicator.o obj/NAZE/core*cm3.o
obj/NAZE/system *stm32f10x.o obj/NAZE/stm32f10x* gpio.o obj/NAZE/stm32f10x*fsmc.o obj/NAZE/stm32f 10x* exti.o
obj/NAZE/stm32f10x*bkp.o obj/NAZE/stm32f10x* spi.o obj/NAZE/stm32f10x_ adc.o obj/NAZE/stm32f10x*iwdg.o*
obj/NAZE/misc.o obj/NAZE/stm32f10x cec.o obj/NAZ E/stm32f10x*wwdg.o obj/NAZE/stm32f10x* tim.o
obj/NAZE/stm32f10x*usart.o obj/NAZE/ stm32f10x* crc.o obj/NAZE/stm32f10x*flash.o obj/NAZE/stm32f10x* rcc.o obj/NAZE/stm
32f10x*sdio.o obj/NAZE/stm32f10x* i2c.o obj/NAZE/stm32f10x*pwr.o obj/NAZE/stm32f1 0x* rtc.o obj/NAZE/stm32f10x*can.o*
obj/NAZE/stm32f10x dac.o obj/NAZE/stm32f10x*db gmcu.o obj/NAZE/stm32f10x* dma.o -lm -mthumb -mcpu=cortex-m3 -static
-Wl,-gc-sect ions,-Map,.//obj/cleanflight*NAZE.map -T.//stm32*flash*f103.ld arm-none-eabi-objcopy -O ihex --set-start*
*0x8000000 obj/cleanflight*NAZE.elf obj /cleanflight_NAZE.hex

cleanflight_NAZE.hex is about 220kb.

Migrating from baseflight

Procedure

First ensure your main flight battery is disconnected or your props are off!

Before flashing with cleanflight, dump your configs for each profile via the CLI and save to a text file.

```
profile 0
dump
```

```
profile 1
dump
profile 2
dump
```

Then after flashing cleanflight paste the output from each dump command into the cli, switching profiles as you go.

You'll note that some commands are not recognised by cleanflight when you do this. For the commands that are not recognised look up the new configuration options and choose appropriate values for the settings. See below for a list of differences.

Once you've done this for the first profile, save the config. Then verify your config is OK, e.g. features serial ports, etc. When you've verified the first profile is OK repeat for the other profiles.

It's also advisable to take screenshots of your AUX settings from baseflight configurator and then after re-applying the settings verify your aux config is correct - some changes were made and some aux settings are not backwards compatible.

CLI command differences from baseflight

In general all CLI commands use underscore characters to separate words for consistency. In baseflight the format of CLI commands is somewhat haphazard.

gps_baudrate

reason: simplify

Cleanflight uses normal baud rate values for gps baudrate, baseflight uses an index.

If an unsupported baud rate value is used the gps code will select 115200 baud.

example: `set gps_baudrate = 115200`

gps_type

reason: renamed to `gps_provider` for consistency

serialrx_type

reason: renamed to `serialrx_provider` for consistency

rssi*aux*channel

reason: improved functionality

Cleanflight supports using any RX channel for rssi. Baseflight only supports AUX1 to 4.

In Cleanflight a value of 0 disables the feature, a higher value indicates the channel number to read RSSI information from.

Example: to use RSSI on AUX1 in Cleanflight use `set rssi_aux_channel = 5`, since 5 is the first AUX channel.

failsafe*detect*threshold

reason: improved functionality

See `failsafe_min_usec` and `failsafe_max_usec` in Failsafe documentation.

emfavoidance

reason: renamed to `emf_avoidance` for consistency

yawrate

reason: renamed to `yaw_rate` for consistency

yawdeadband

reason: renamed to yaw_deadband for consistency

midrc

reason: renamed to `midrc` for consistency

mincheck

reason: renamed to `min_check` for consistency

maxcheck

reason: renamed to `max_check` for consistency

minthrottle

reason: renamed to `min_throttle` for consistency

maxthrottle

reason: renamed to `max_throttle` for consistency

mincommand

reason: renamed to `min_command` for consistency

deadband3d_low

reason: renamed to `3d_deadband_low` for consistency

deadband3d_high

reason: renamed to `3d_deadband_high` for consistency

deadband3d_throttle

reason: renamed to `3d_deadband_throttle` for consistency

neutral3d

reason: renamed to `3d_neutral` for consistency

alt*hold*throttle_neutral

reason: renamed to 'alt*hold*deadband'

GPS

Two GPS protocols are supported. NMEA text and UBLOX Binary.

GPS Provider

Set the `gps_provider` appropriately.

Value	Meaning
0	NMEA
1	UBLOX

SBAS

When using a UBLOX GPS the SBAS mode can be configured using `gps_sbas_mode`.

The default is AUTO.

Value	Meaning	Region
0	AUTO	Global

1	EGNOS	Europe
2	WAAS	North America
3	MSAS	Asia
4	GAGAN	India

If you use a regional specific setting you may achieve a faster GPS lock than using AUTO.

Autotune

Autotune helps to automatically tune your multirotor.

Configuration.

Autotune only works in HORIZON or ANGLE mode, before using auto-tune it's best you setup so there is as little drift as possible. Autotuning is best on a full battery in good flying conditions, i.e. no or minimal wind.

Configure a two position switch on your transmitter to activate the AUTOTUNE and HORIZON or ANGLE modes using the auxilary configuration. You may find a momentary switch more suitable than a toggle switch.

Using autotuning

Turn off the autotune switch. If the autotune switch is on while not armed the warning LED will flash and you cannot arm.

Launch the multirotor.

Turn on/hold the autotune switch on your transmitter.

Observe roll left/right. A beep code will sound on the beeper.

Turn off/release the switch while still flying to stop this phase of tuning.

PID settings will have been updated for ROLL/YAW.

Turn on/hold the switch again.

Observe pitch forwards/backwards. A beep code will sound on the beeper.

Turn off/release the switch while still flying to stop this phase of tuning.

PID settings will have been updated for PITCH/YAW.

PIDS are updated, fly and see if it's better.

If it's worse, flip the switch again to restore previous pids that were present prior to arming.

Land.

Verify results via an app while power still applied if desired.

Cutting the power will loose the unsaved pids.

If you're happy with the pids then while disarmed flip the autotune switch again to save all settings then flip it back so you can arm again.

A beeper will sound indicating the settings are saved.

References

- Brad Quick for the initial Autotune algorithm in BradWii.# Display

Cleanflight supports displays to provide information to you about your aircraft and cleanflight state.

When the aircraft is armed the display does not update so flight is not affected. When disarmed the display cycles between various pages.

There is currently no way to change the information on the pages, the list of pages or the time between pages - Code submissions via pull-requests are welcomed!

Supported Hardware

At this time no other displays are supported other than the SSD1306 / UG-2864HSWEG01.

Configuration

From the CLI enable the `DISPLAY` feature

```
feature DISPLAY
```

SSD1306 OLED displays

The SSD1306 display is a 128x64 OLED display that is visible in full sunlight, small and consumes very little current. This makes it ideal for aircraft use.

There are various models of SSD1306 boards out there, they are not all equal and some require addtional modifications before they work. Choose wisely!

Links to screens:

http://www.banggood.com/0_96-Inch-I2C-IIC-SPI-Serial-128-X-64-OLED-LCD-LED-Display-Module-p-922246.html
http://www.wide.hk/products.php?product=I2C-0.96%22-OLED-display-module-%28-compatible-Arduino-%29
http://witespyquad.gostorego.com/accessories/readytofly-1-oled-128x64-pid-tuning-display-i2c.html
http://www.multiwiicopter.com/products/1-oled

The banggood.com screen is the cheapest at the time fo writing and will correctly send I2C ACK signals.

Crius CO-16

This screen is best avoided but will work if you modify it.

Step 1

As supplied the I2C ack signal is not sent because the manufacturer did not bridge D1 and D2 together. To fix this solder the two pins together as they enter the screen. Failure to do this will result is a screen that doesn't display anything.

Step 2

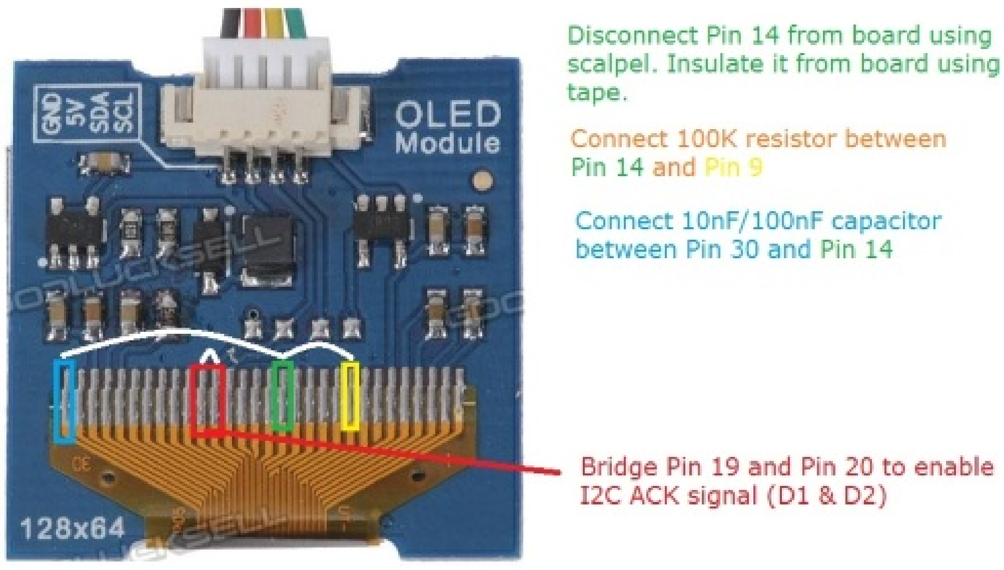
Pin 14 must be disconnected from the main board using a scalpel. Then connect a 10nF or 100nF capacitor between pins 30 and the lifted pin 14.

Step 3

Connect a 100K resistor between Pin 9 and the lifted Pin 14.

Failure to perform steps 2 and 3 will result in a display that only works on power up some of the time any may display random dots or other display corruption.

More can be read about this procedure here: http://www.multiwii.com/forum/viewtopic.php?f=6&t=2705&start=10


GND
5V
SDA
SCL
OLED Module
128x64
Disconnect Pin 14 from board using scalpel. Insulate it from board using tape.
Connect 100K resistor between Pin 14 and Pin 9
Connect 10nF/100nF capacitor between Pin 30 and Pin 14
Bridge Pin 19 and Pin 20 to enable I2C ACK signal (D1 & D2)

Connections

Connect +5v, Ground, I2C SDA and I2C SCL from the flight controller to the display.

On Naze32 rev 5 boards the SDA and SCL pins are underneath the board.

Development

This document is primarily for developers only.

Unit testing

Ideally, there should be tests for any new code. However, since this is a legacy codebase which was not designed to be tested this might be a bit difficult.

If you want to make changes and want to make sure it's tested then focus on the minimal set of changes required to add a test.

Tests currently live in the `test` folder and they use the google test framework.
The tests are compiled and run natively on your development machine and not on the target platform. This allows you to develop tests and code and actually execute it to make sure it works without needing a development board or simulator.

This project could really do with some functional tests which test the behaviour of the application.

All pull requests to add/improve the testability of the code or testing methods are highly sought!

General principals

1. Name everything well.
2. Strike a balance between simplicity and not-repeating code.
3. Methods that return a boolean should be named as a question, and should not change state. e.g. 'isOkToArm()'
4. Methods that start with the word 'find' can return a null, methods that start with 'get' should not.
5. Methods should have verb or verb-phrase names, like `deletePage` or save. Variables should not, they generally should be nouns. Tell the system to 'do' something 'with' something. e.g. deleteAllPages(pageList).
6. Keep methods short - it makes it easier to test.
7. Don't be afraid of moving code to a new file - it helps to reduce test dependencies.
8. Avoid noise-words in variable names, like 'data' or 'info'. Think about what you're naming and name it well. Don't be afraid to rename anything.
9. Avoid comments taht describe what the code is doing, the code should describe itself. Comments are useful however for big-picture purposes and to document content of variables.
10. If you need to document a variable do it at the declarion, don't copy the comment to the `extern` usage since it will lead to comment rot.
11. Seek advice from other developers - know you can always learn more.
12. Be professional - attempts at humor or slating existing code in the codebase itself is not helpful when you have to change/fix it.
13. Know that there's always more than one way to do something and that code is never final - but it does have to work.

Running the tests.

The tests and test build system is very simple and based of the googletest example files, it will be improved in due course.

```
cd test
make
```

This will build a set of executable files, one for each `*_unittest.cc` file.

You can run them on the command line to execute the tests and to see the test report.

You can also step-debug the tests in eclipse and you can use the GoogleTest test runner to make building and re-running the tests simple.

The tests are currently always compiled with debugging information enabled, there may be additional warnings, if you see any warnings please attempt to fix them and submit pull requests with the fixes.

TODO

- Test OpenLRSNG's RSSI PWM on AUX5-8.
- Add support for UART3/4 on STM32F3.
- Cleanup validateAndFixConfig and pwm_mapping.c to use some kind of feature/timer/io pin mapping to remove #ifdef
- Split RX config into RC config and RX config.
- Enabling/disabling features should not take effect until reboot since. Main loop executes and uses new flags as they are set in the cli but appropriate init methods will not have been called which results in undefined behaviour and could damage connected devices - this is a legacy problem from baseflight.
- Solve all the naze rev4/5 HSE*VALUE == 8000000/1200000 checking, the checks should only apply to the naze32 target. See system* stm32f10x.c/SetSysClock().

Known Issues

- Softserial RX on STM32F3 does not work. TX is fine.
- Dynamic throttle PID does not work with new pid controller.
- Autotune does not work yet with with new pid controller.

RSSI

RSSI is a measurement of signal strength. RSSI is very handy so you know when you are going out of range or there is interference.

Some receivers have RSSI outputs. 3 types are supported.

1. RSSI via PPM channel
2. RSSI via Parallel PWM channel
3. RSSI via ADC with PPM RC that has an RSSI output - aka RSSI ADC

RSSI via PPM

Configure your receiver to output RSSI on a spare channel, then select the channel used via the cli.

e.g. if you used channel 1 then you would set:

```
set rssi_channel = 1
```

RSSI via Parallel PWM channel

Connect the RSSI signal to any PWM input channel then set the RSSI channel as you would for RSSI via PPM

RSSI ADC

Connect the RSSI signal to the RC2/CH2 input. The signal must be between 0v and 3.3v to indicate between 0% and 100% RSSI. Use inline resistors to lower voltage if required, inline smoothing capacitors may also help.

FrSky D4R-II and X8R supported.

Enable using the RSSI_ADC feature:

```
feature RSSI_ADC
```

The feature can not be used when RX*PARALLEL*PWM is enabled.

Buzzer

Cleanflight supports a buzzer which is used for the following purposes, and more:

Low Battery alarm (when battery monitoring enabled) Notification of calibration complete status. AUX operated beeping - useful for locating your aircraft after a crash. Failsafe status.

Buzzer is enabled by default on platforms that have buzzer connections.

Types of buzzer supported

The buzzers are enabled/disabled by simply enabling or disabling a GPIO output pin on the board. This means the buzzer must be able to generate it's own tone simply by having power applied to it.

Buzzers that need an analogue or PWM signal do not work and will make clicking noises or no sound at all.

Example of a known-working buzzer.

http://www.rapidonline.com/Audio-Visual/Hcm1205x-Miniature-Buzzer-5v-35-0055

Connections

Naze32

Connect a supported buzzer directly to the BUZZ pins. Observe polarity.

CC3D

Buzzer support on the CC3D requires that a buzzer circuit be created to which the input is PA15. PA15 is unused and not connected according to the CC3D Revision A schematic. Connecting to PA15 requires careful soldering.

See the CC3D - buzzer circuir.pdf for details.

Serial

Cleanflight has enhanced serial port flexibility but configuration is slightly more complex as a result.

Cleanflight has the concept of a function (MSP, GPS, Serial RX, etc) and a port (VCP, UARTx, SoftSerial x). Not all functions can be used on all ports due to hardware pin mapping, conflicting features, hardware and software constraints.

Serial port types

- USB Virtual Com Port (VCP) - USB pins on a USB port connected directly to the processor without requiring a dedicated USB to UART adapter. VCP does not 'use' a physical UART port.
- UART - A pair of dedicated hardware transmit and receive pins with signal detection and generation done in hardware.
- SoftSerial - A pair of hardware transmit and receive pins with signal detection and generation done in software.

UART is the most efficent in terms of CPU usage. SoftSerial is the least efficient and slowest, softserial should only be used for low-bandwith usages, such as telemetry transmission.

UART ports are sometimes exposed via on-board USB to UART converters, such as the CP2102 as found on the Naze and Flip32 boards. If the flight controller does not have an onboard USB to UART converter and doesn't support VCP then an external USB to UART board is required. These are sometimes referred to as FTDI boards. FTDI is just a common manufacter of a chip (the FT232RL) used on USB to UART boards.

When selecting a USB to UART converter choose one that has DTR exposed as well as a selector for 3.3v and 5v since they are more useful.

Examples: http://www.banggood.com/FT232RL-FTDI-USB-To-TTL-Serial-Converter-Adapter-Module-For-Arduino-p-917226.html http://www.banggood.com/Wholesale-USB-To-TTL-Or-COM-Converter-Module-Buildin-in-CP2102-New-p-27989.html

Both SoftSerial and UART ports can be connected to your computer via USB to UART converter boards.

Serial Configuration

To make configuration easier common usage scenarios are listed below.

Serial port scenarios

```
0   UNUSED
1   MSP, CLI, TELEMETRY, GPS-PASTHROUGH
2   GPS ONLY
3   RX SERIAL ONLY
4   TELEMETRY ONLY
5   MSP, CLI, GPS-PASTHROUGH
6   CLI ONLY
7   GPS-PASSTHROUGH ONLY
8   MSP ONLY
```

Contraints

- There must always be a port available to use for MSP
- There must always be a port available to use for CLI
- To use a port for a function, the function's corresponding feature must be enabled first. e.g. to use GPS enable the GPS feature.
- If the configuration is invalid the serial port configuration will reset to it's defaults and features may be disabled.

Examples

All examples assume default configuration (via cli `defaults` command)

a) GPS and FrSky TELEMETRY (when armed)

- TELEMETRY,MSP,CLI,GPS PASSTHROUGH on UART1
- GPS on UART2

```
feature TELEMETRY
feature GPS
save
```

b) RX SERIAL and FrSky TELEMETRY (when armed)

- TELEMETRY,MSP,CLI,GPS PASSTHROUGH on UART1
- RX SERIAL on UART2

```
feature -RX_PARALLEL_PWM
feature RX_SERIAL
feature TELEMETRY
set serial_port_2_scenario = 3
save
```

b) RX SERIAL and FrSky TELEMETRY via softserial

- MSP,CLI,GPS PASSTHROUGH on UART1
- RX SERIAL on UART2
- TELEMETRY on SOFTSERIAL1

```
feature -RX_PARALLEL_PWM
feature RX_SERIAL
feature TELEMETRY
feature SOFTSERIAL
set serial_port_2_scenario = 3
set serial_port_3_scenario = 4
save
```

c) GPS and FrSky TELEMETRY via softserial

- MSP,CLI,GPS PASSTHROUGH on UART1
- GPS on UART2
- TELEMETRY on SOFTSERIAL1

```
feature -RX_PARALLEL_PWM
feature RX_PPM
feature TELEMETRY
feature GPS
feature SOFTSERIAL
set serial_port_3_scenario = 4
save
```

d) RX SERIAL, GPS and TELEMETRY (when armed) MSP/CLI via softserial

- GPS on UART1
- RX SERIAL on UART2
- TELEMETRY,MSP,CLI,GPS PASSTHROUGH on SOFTSERIAL1

```
feature -RX_PARALLEL_PWM
feature TELEMETRY
feature GPS
feature RX_SERIAL
feature SOFTSERIAL
set serial_port_1_scenario = 2
set serial_port_2_scenario = 3
set serial_port_3_scenario = 1
set msp_baudrate = 19200
set cli_baudrate = 19200
set gps_passthrough_baudrate = 19200
save
```

e) HoTT Telemetry via UART2

- MSP,CLI,GPS PASSTHROUGH on UART1
- HoTT telemetry on UART2

```
feature -RX_PARALLEL_PWM
feature RX_PPM
feature TELEMETRY
set serial_port_2_scenario = 4
set telemetry_provider = 1
```

f) GPS, HoTT Telemetry via SoftSerial 1

- MSP,CLI,GPS PASSTHROUGH on UART1
- GPS on UART2
- HoTT telemetry on SOFTSERIAL1

```
feature -RX_PARALLEL_PWM
feature RX_PPM
feature TELEMETRY
feature GPS
feature SOFTSERIAL
set serial_port_3_scenario = 4
set telemetry_provider = 1
save
```

Telemetry

Telemetry allows you to know what is happening on your aircraft while you are flying it. Among other things you can receive battery voltages and GPS positions on your transmitter.

Telemetry can be either always on, or enabled when armed. If no serial port is set to be telemetry-only then telemetry will only be enabled when armed.

Telemetry is enabled using the 'TELEMETRY` feature.

```
feature TELEMETRY
```

Three telemetry providers are currently supported, FrSky (the default), Graupner HoTT V4 and MultiWii Serial Protocol (MSP)

Use the `telemetry_provider` cli command to select one.

Value	Meaning
0	FrSky (Default)
1	HoTT
2	MSP

Example:

```
set telemetry_provider = 1
```

There are further examples in the Configuration section of the documentation.

FrSky telemetry

FrSky telemetry is transmit only and just requires a single connection from the TX pin of a serial port to the RX pin on an FrSky telemetry receiver.

FrSky telemetry signals are inverted. To connect a cleanflight capable board to an FrSKy receiver you have some options.

1. A hardware inverter - Built in to some flight controllers.
2. Use software serial and enable frsky_inversion.
3. Use a flight controller that has software configurable hardware inversion (e.g. STM32F30x).

For 1, just connect your inverter to a usart or software serial port.

For 2 and 3 use the cli command as follows:

```
set frsky_inversion = 1
```

HoTT telemetry

HoTT telemetry can be used when the TX and RX pins of a serial port are connected using a diode and a single wire to the T port on a HoTT receiver.

Only Electric Air Modules and GPS Modules are emulated, remember to enable them on your transmitter - in the Telemetry Menu on the MX-20.

Serial ports use two wires but HoTT uses a single wire so some electronics are required so that the signals don't get mixed up.

Connect as follows:

```
HoTT TX/RX -> Serial RX (connect directly)
Serial TX -> 1N4148 Diode -(|  )-> HoTT TX/RX (connect via diode)
```

The diode should be arranged to allow the data signals to flow the right way

```
-(|  )- == Diode, | indicates cathode marker.
```

As noticed by Skrebber the GR-12 (and probably GR-16/24, too) are based on a PIC 24FJ64GA-002, which has 5V tolerant digital pins.

Note: The softserial ports are not listed as 5V tolerant in the STM32F103xx data sheet pinouts and pin description section. Verify if you require a 5v/3.3v level shifters.

MultiWii Serial Protocol (MSP)

MSP Telemetry simply transmitts MSP packets in sequence to any MSP device attached to the telemetry port. It rotates though a fixes sequence of command responses.

It is transmit only, it can work at any supported baud rate.

MSP telemetry is currently only output on serial ports that are set to MSP, NOT telemetry. This will likely change in the future so that the MSP telemetry uses ports configured as telemetry just like the other providers do.

Battery Monitoring

Cleanflight has battery monitoring capability. Battery voltage of the main battery can be measured by the system and used to trigger a low-battery warning buzzer, on-board status LED flashing and LED strip patterns.

Low battery warnings can:

- help to ensure that you have time to safely land the aircraft.
- help maintain the life and safety of your LiPo/LiFe batteries which should not be discharged below manufactures recommendations.

Minimum and maximum cell voltages can be set, these voltages are used to detect the amount of cells you are using.

Per-cell monitoring is not supported.

Supported targets

All targets support battery voltage monitoring unless status.

Connections

When dealing with batteries ALWAYS CHECK POLARITY!

Measure expected voltages first and then connect to flight controller, connecting to the flight controller with incorrect voltage or reversed polarity will likely fry your flight controller.

Naze32

The Naze32 has an on-board battery divider circuit, connect your main battery to the VBAT connector.

CC3D

The CC3D has no battery divider, create one that gives you a 3.3v MAXIMUM output when your battery is fully charged and connect the output from it to S5_IN/PA0/RC5.

S5_IN/PA0/RC5 is Pin 7 on the 8 pin connector, second to last pin, opposite end from the GND/+5/PPM signal input.

Note: When battery monitoring is enabled on the CC3D RC5 can no-longer be used for PWM input.

Configuration

Enable the VBAT feature.

Configure min/max cell voltages using the following CLI commands:

`vbat_scale` - adjust this to match battery voltage to reported value.

`vbat_max_cell_voltage` - maximum voltage per cell, used for auto-detecting battery voltage in 0.1V units, i.e. 43 = 4.3V

`vbat_min_cell_voltage` - minimum voltage per cell, this triggers battery out alarms, in 0.1V units, i.e. 33 = 3.3V

e.g.

```
set vbat_scale = 110
set vbat_max_cell_voltage = 43
set vbat_min_cell_voltage = 33
```