Pinio & PinioBox

Overview

- PINIO is an abstraction of simple GPIO (General Purpose I/O) pin facility, and PINIO BOX is a facility to associate boxes (modes) to PINIO.
- Most targets are build to be capable of configuring up to four pins for the PINIO facility, which can then be controlled by the PINIO BOX facility.

PINIO

An MCU pin can be assigned to `PINIO` by `resource` CLI command.

```
resource PINIO <index> <pinID>
```

Example of `resource` with `PINIO`:

```
[...]
resource PINIO 1 A01
resource PINIO 2 A08
resource PINIO 3 C09
resource PINIO 4 D02
[...]
```

PINIO CONFIG

I/O configuration of each pin is specified by `pinio_config` CLI variable, which is a comma separated list/array of 8-bit values. MSB represents inversion, and remaining 7-bits specify I/O mode as defined in drivers/pinio.h (Only push-pull output is defined for this PR).

PINIO CONFIG	HEX	DEC
PINIO_CONFIG_OUT_INVERTED	0x80	128
PINIO_CONFIG_MODE_MASK	0x7F	127

PINIO CONFIG	HEX	DEC
PINIO_CONFIG_MODE_OUT_PP	0x01	1

`Note`: Values can be combined together

Example of `pinio_config`

```
set pinio_config = 1,129,1,1
```

Sets Output-Push-Pull for PINIO #1, #3 and #4, and Inverted Output-Push-Pull for PINIO #2.

Default value is `1` (Output-Push-Pull).

PINIO BOX

CLI variable `pinio_box`, comma separated list of permanent ID of boxes, associate the boxes to corresponding PINIOs. Once associated, the boxes's activation status are reflected to the associated PINIOs (and then to pins). Since the PINIO BOX facility has it's own capability of monitoring activation state of boxes, it operates independently from what boxes are meant for. (In other words, the PINIO BOX **adds** PINIO capability to boxes.)

Permanent IDs `40` through `43` are user defined boxes, which are activated and appear as USER1 through USER4 in the list of boxes at places such as Modes tab in configurator.

Example of `pinio_box`

```
set pinio_box = 0, 39, 43, 255
```

With this assignment, PINIO #1 through #4 are associated with boxes as follow.

`PINIO`	Description	Permanent ID
`1`	`ARM`	`0`
`2`	`VTX PIT MODE`	`39`
`3`	`USER4`	`43`

`PINIO`	Description	Permanent ID
4	`BOXID_NONE`	255

For permanent ID of boxes (or AUX modes), the table below is based on msp/msp_box.c.

Box	Mode	ID	Notes
`BOXARM`	ARM	0	
`BOXANGLE`	ANGLE	1	
`BOXHORIZON`	HORIZON	2	
`BOXALTHOLD`	ALTITUDE HOLD	3	2025.12
`BOXANTIGRAVITY`	ANTI GRAVITY	4	
`BOXMAG`	MAG	5	
`BOXHEADFREE`	HEADFREE	6	
`BOXHEADADJ`	HEADADJ	7	
`BOXCAMSTAB`	CAMSTAB	8	
`BOXCAMTRIG`	CAMTRIG	9	removed
`BOXGPSHOME`	GPS HOME	10	removed
`BOXPOSHOLD`	POSITION HOLD	11	2025.12
`BOXPASSTHRU`	PASSTHRU	12	
`BOXBEEPERON`	BEEPER	13	
`BOXLEDMAX`	LEDMAX	14	removed
`BOXLEDLOW`	LEDLOW	15	

Box	Mode	ID	Notes
`BOXLLIGHTS`	LLIGHTS	`16`	removed
`BOXCALIB`	CALIB	`17`	
`BOXGOV`	GOVERNOR	`18`	removed
`BOXOSD`	OSD DISABLE SW	`19`	
`BOXTELEMETRY`	TELEMETRY	`20`	
`BOXGTUNE`	GTUNE	`21`	removed
`BOXRANGEFINDER`	RANGEFINDER	`22`	removed
`BOXSERVO1`	SERVO1	`23`	
`BOXSERVO2`	SERVO2	`24`	
`BOXSERVO3`	SERVO3	`25`	
`BOXBLACKBOX`	BLACKBOX	`26`	
`BOXFAILSAFE`	FAILSAFE	`27`	
`BOXAIRMODE`	AIR MODE	`28`	
`BOX3D`	DISABLE / SWITCH 3D	`29`	
`BOXFPVANGLEMIX`	FPV ANGLE MIX	`30`	
`BOXBLACKBOXERASE`	BLACKBOX ERASE (>30s)	`31`	
`BOXCAMERA1`	CAMERA CONTROL 1	`32`	
`BOXCAMERA2`	CAMERA CONTROL 2	`33`	
`BOXCAMERA3`	CAMERA CONTROL 3	`34`	
`BOXFLIPOVERAFTERCRASH`	FLIP OVER AFTER CRASH	`35`	

Box	Mode	ID	Notes
`BOXPREARM`	PREARM	`36`	
`BOXBEEPGPSCOUNT`	BEEP GPS SATELLITE COUNT	`37`	
`BOX3DONASWITCH`	3D ON A SWITCH	`38`	removed
`BOXVTXPITMODE`	VTX PIT MODE	`39`	
`BOXUSER1`	USER1	`40`	
`BOXUSER2`	USER2	`41`	
`BOXUSER3`	USER3	`42`	
`BOXUSER4`	USER4	`43`	
`BOXPIDAUDIO`	PID AUDIO	`44`	
`BOXPARALYZE`	PARALYZE	`45`	
`BOXGPSRESCUE`	GPS RESCUE	`46`	
`BOXACROTRAINER`	ACRO TRAINER	`47`	
`BOXVTXCONTROLDISABLE`	DISABLE VTX CONTROL	`48`	
`BOXLAUNCHCONTROL`	LAUNCH CONTROL	`49`	
`BOXMSPOVERRIDE`	MSP OVERRIDE	`50`	

`Note`: Value 255 is defined as BOXID_NONE which specifies the specified slot is not used.

Examples

These are some **general** examples.

Some inbuilt or external devices/modules come with a dedicated pin/wire to enable/disable it. This depends on the device, but such a pin/wire is *necessary* for the PINIO function to work the way it is supposed to. On some devices this pin needs to be set `HIGH/ON/1` (about 3,3V) to enable it, on others, it needs to be `LOW/OFF/0` (about 0 V) to do the same. This really depends on the device.

Find already defined PINIO functions

Some flight controller targets come with PINIOs *preconfigured*, for example to switch the camera, the VTX supply voltage or an inbuilt Bluetooth device.

Check the unified target configuration of **your** flight controller or use the `resource` command on the running board, to find any existing PINIOs -- if any.

You may find something like `resource PINIO 1 B00` or `PINIO 1 B00`. This shows us, that there is already a PINIO with the index of `1` (PINIO #1).

If you want to add additional PINIOs, **make sure not to remove existing ones in the process, by incrementing the index!**

Disable internal Bluetooth device, when craft gets armed

Most flight controller targets with an inbuilt Bluetooth module, are already *preconfigured*, so there is often no need to do anything at all!

Still, it may be interesting to find out, if and how it is configured.

The configuration may look like this:

```
# resource
[...]
resource PINIO 1 B00
[...]

# get pinio_config
pinio_config = 129,1,1,1
Array length: 4

# get pinio_box
pinio_box = 0,255,255,255
Array length: 4
Default value: 0,255,255,255
```

This is PINIO #1(mapped to pin `B00`), so we only look at the *first* of the four comma separated values in the arrays.

The value of `pinio_box` is `0`, which -- according to the table above -- is set to the `BOXARM` box/mode. So if the craft gets armed, this gets activated, too.

The value of `pinio_config` is `129` which -- very simply put -- means it is set to on/off mode *and* also gets inverted. *Inverted* means, is gets turned off, when the matching PINIO gets turned on -- and vice versa.

So if the craft gets armed, the `B00` pin will be set to.... `LOW` (about 0 V). In this example, this means, that the inbuilt Bluetooth module turns off, when it senses a LOW signal on this pin (simply because it's built this way).

Disable external Bluetooth UART-Adapter, when the craft gets armed

This is bascially the same as with the internal module.

The only difference is, that you need to select a free pin to switch the device on and off and you need to know what type of signal (HIGH or LOW) your device expects.

Let's say you decide to use the otherwise unused PPM pin to switch your newly added Bluetooth module, which comes with an additional pin/wire that may be labeled POWER or ENABLE. The documentation of the module tells you, that the signal on this pin needs to be LOW for the module to turn off. Please note, that there *maybe* pins on the breakout board of the Bluetooth module that aren't actually connected to anything!

You also need to know if you already have on or more `PINIO` functions configured (see above!).

```
# resource
[...]
resource PPM B09
[...]
resource PINIO 1 C08
resource PINIO 2 C09
[...]

# resource PPM none
Resource is freed

# resource PINIO 3 B09
Resource is set to B09

# get pinio_config
pinio_config = 1,1,1,1
Array length: 4

# set pinio_config = 1,1,129,1
pinio_config set to 1,1,129,1

# get pinio_box
pinio_box = 40,41,255,255

# set pinio_box = 40,41,0,255
pinio_box set to 40,41,0,255

# save
```

Some things to note here:

- There are already *two* PINIO functions set up, so the one we add will be #3.
- We set the *third* value of `pinio_config` from `1` to `129` (inversion), because the device shall switch *off* when this box/mode turns *on*.
- We set the *third* value of `pinio_box` from `255` to `0` because we want to activate this, when the device is armed (see table above!).
- Don't forget to save!

Switch Bluetooth device on, when in Pit Mode

This is the exact same setup as above, but instead of setting `pinio_box` for PINIO #3 to `0`, it is set to `39`. Now the Bluetooth module will turn on, when in pit mode.

Switch VTX on/off using a USERn function controlled by the sender

This time, we want to switch the VTX with an actual switch on the sender. For this to work, we need

- A VTX that comes with a pin to switch it off.
- A function on the flight controller board that allows to switch off the supply voltage for the VTX.
- A simple circuit with a transistor, that gets placed between VTX supply voltage and VTX. There are ready made modules like "RealPit" available for this.

For this, we use one of the `BOXUSER<n>` modes/boxes (IDs 40 to 43). This will add a matching User mode/function in the Modes tab of the Configurator, which can then be mapped to a channel on the sender.

Let's say we have a "clean" flight controller without any pre-configured PINIOs or User functions and the PPM pin is the one to be used for this:

```
#resource
[...]
resource PPM B03
[...]

# resource PPM none
Resource is freed

# resource PINIO 1 B03
Resource is set to B03

# get pinio_box
pinio_box = 255,255,255,255
```

```
# set pinio_box = 40,255,255,255
pinio_box set to 40,255,255,255

# save
```

Some things to note here:

- We don't change `pinio_config`, since the example VTX will switch *on* when the pin is pulled HIGH. So the default value of `1` is fine.
- We set the first value of `pinio_box` to `40`, since this is the first and only User function on this board (again, see table above!).
- The only thing left is now to map the function in the Modes tab of the Configurator (not, that this will only show up, after setting an ID of `40-43` in `pinio_box`).
- Don't forget to save!

FAQ

- None yet.

Youtube

Joshua Bardwell

Ivan Efimov

Links

Oscar Liang

VitroidFPV

Community

Discord

IntoFPV

Feeds

RSS

Atom

JSON