FluxGrip FG40 datasheet

Revision 2024-11-29

Zubax Robotics
Uus 12, Tallinn 10111, Estonia
info@zubax.com
Q&A: forum.zubax.com

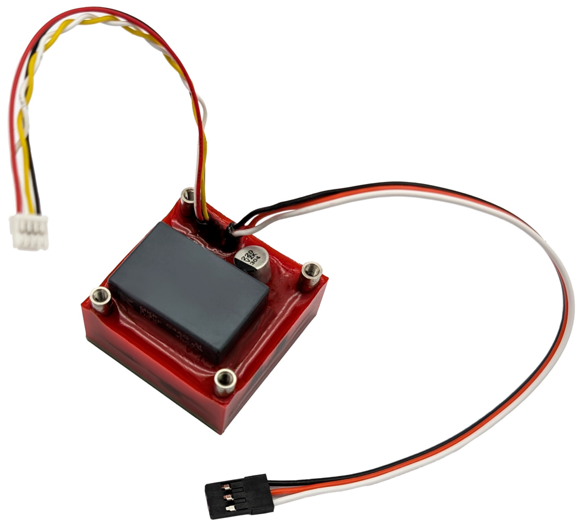
Overview

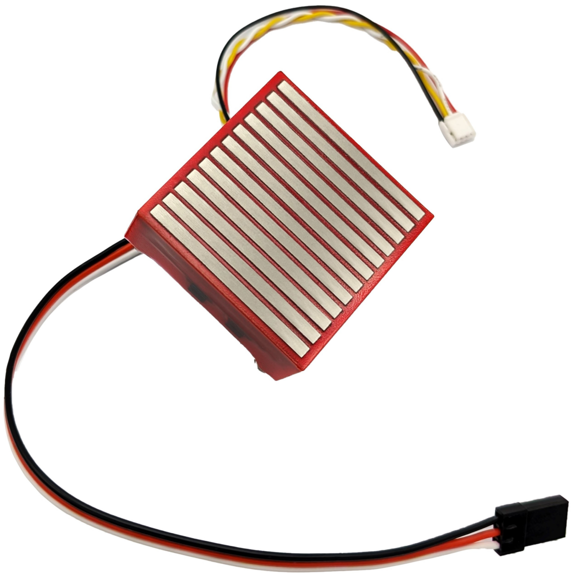
FluxGrip FG40 is an *electro-permanent magnet*. It combines the advantages of electro- and permanent magnets by being able to switch between the on-state and off-state on demand (like an electromagnet) while consuming zero power in either state (like a permanent magnet). The design is entirely solid-state, with no moving parts inside.

With a wide supply voltage range and multiple communication interfaces, the device easily integrates into any end system.

FluxGrip adheres to safety and reliability standards that ensure dependable operation in hostile environments.

Certain features are available only in specific models (section 1.1).





Applications

- Payload attachment in unmanned vehicles (UAV, AUV, etc.).
- Workholding in CNC machines, manipulators, and robots.
- Magnetic locks, clamps, and holders.

Features

- Strong magnet in a very compact and lightweight form-factor.
- Zero power consumption in either on or off state.
- Fast on↔off state transition.
- Low (de-)magnetization energy consumption.
- Entirely solid-state construction, no moving parts inside.
- IP69 rated — dust-tight, submersible.
- AEC-Q grade 2 electronics.
- Resistant to vibration, mechanical stress, high and low temperatures, electromagnetic interference, humidity, dust, and chemically aggressive environments.
- Wide power supply voltage range ensures trivial integration into the end system.
- Compatible with industry-standard interfaces:
  - Cyphal/CAN FD control with diagnostic telemetry;
  - RC PWM control input;
  - voltage level control input;
  - pull resistor analog status feedback.

Table of contents

List of tables

List of figures

1 Functional description

Warning: High voltage. Do not attempt to disassemble the device to avoid the risk of electric shock.

FG40 is an electropermanent magnet that contains a remagnetizable magnetic assembly and the control electronics in a single unit. The magnetic assembly can be magnetized, which is referred to as the *on-state*, and demagnetized or relaxed, which is referred to as the *off-state*. Both states are stable, allowing the device to remain in either state indefinitely when powered down; external power is only required for state transitions.

To transition between the on and off states, the device consumes a small amount of energy from the power supply input. The device is designed to consume a fixed input current, which implies that the power consumption increases with the supply voltage, and therefore, the time it takes to transition between the states is inversely proportional to the supply voltage. The magnetization and relaxation processes follow a non-linear time profile, with most of the transition effect occurring rapidly relative to the total transition time.

When the device is powered on but does not transition between the on and off states — or, in other words, resides in the *idle state* — some insignificant idle power is consumed by the built-in electronics and its communication interfaces. It is possible to power down the device completely when switching is not needed to conserve energy further.

The force that is required to pull the payload away from the magnet — the *holding force* — depends not only on the performance of the magnet itself, but also on the qualities of the payload. Materials with high magnetic permeability ($\mu_r > 4000$), such as iron, electrical steel, or specialized magnetic alloys like permendur, are recommended for the payload. The payload needs to be able to accommodate the magnetic flux maintained by the magnet without saturating; otherwise, the flux and thus the holding force will be limited by the magnetic saturation of the payload material.

Another factor affecting the holding force is the proximity of the payload to the magnet poles. For optimal performance, the payload should be in direct contact with the magnet poles across its entire area. Imperfect contact, foreign objects, dirt, dust, or metallic shavings between the magnet and the payload may impair the performance of the magnet and may even cause damage if any hard impurities (such as metallic shavings) are forced into the polymer composite body of the magnetic assembly.

A spurious partial magnetization or relaxation may occur if the magnetic assembly of the device is exposed to a strong external magnetic field. For example, placing a strong permanent magnet directly against the magnetic poles of the device may cause it to partially acquire the magnetization of the external magnet. This effect will be undone at the next magnetization or relaxation cycle.

The magnetic assembly is only able to retain full magnetization as long as it is held in contact with the payload. As soon as the payload is removed, the intrinsic demagnetizing field[1] of the magnetic assembly will cause partial relaxation. This means that once the payload is re-applied, the holding force will be lower than it was before the removal. To restore the full holding force, a magnetization cycle is required; this can be achieved by sending a forced magnetization command to the device as explained below, or, alternatively, by simply cycling the magnet off and then back on again.

The device accepts commands via the communication interfaces in the form of the desired state: OFF, ON, and FORCE, whose handling is described in the following state transition table.

Command	State	Action
OFF	Off	
OFF	On	Demagnetize
ON	Off	Magnetize
ON	On	
FORCE	(any)	Magnetize once

Table 1.1: Magnet state transition

Having received a FORCE command, the device will execute one magnetization cycle regardless of the cur-

[1] https://en.wikipedia.org/wiki/Demagnetizing_field

rent state and ignore further FORCE commands until any other command is received.

Reach out to https://forum.zubax.com or email support@zubax.com for assistance and advice.

1.1 Models

Modifications that are not explicitly listed may be available upon request. Contact Zubax Robotics for details.

Model name	CAN interface	Analog interface
FG401M	x1 UCANPHY Micro	no
FG401MA	x1 UCANPHY Micro	yes
FG402M	x2 UCANPHY Micro	no

Table 1.2: Model comparison

The UCANPHY Specification is available[2] from the OpenCyphal project.

1.2 Interface and power supply

Commands are delivered to the device via any of its communication interfaces, which are:

- Cyphal/CAN[3] (with redundancy as an option for high-integrity systems) — section 3.
- RC PWM (only in some models) — section 4.1.
- DC voltage (only in some models) — section 4.2.

Telemetry data published via Cyphal provides insights into device state and performance — essential for high-integrity systems. Additionally, minimal status reporting is implemented via the pull up/down resistor connected to the analog input port for applications that are unable to leverage the Cyphal interface.

The device is powered either via the Cyphal network ports or via the analog port connector. The built-in voltage regulator accepts a wide range of power supply voltages which allow easy integration of the magnet into the end application without the need for additional power converters. If the supply voltage exceeds the operating limits specified in table 1.3, the device may continue operating but will reject magnetization commands until the voltage returns within limits.

When the device is operating near the minimum supply voltage with a high-impedance power source, an attempt to (de)magnetize may fail due to the voltage drop caused by an increased current consumption. The device will refuse to perform a state transition until the supply voltage is restored to the nominal level (see table 1.3). If the power supply voltage is not restored in a reasonable time since the arrival of the state transition command, the device may enter the fault state (see section 1.3).

The magnet poles and mounting standoffs connect to the power supply ground through a high-impedance parallel RC network, preventing static charge accumulation. The RC network may cause negligible AC leakage currents to flow through the payload and mounting standoffs, which is normal and does not affect the device's operation.

LED status and activity indication is covered in chapter 2.

Parameter	Note	Min	Typ	Max	Unit
Supply voltage		4		30	V
Idle power consumption	magnet not switching		50		mW
Switching current consumption	CC mode, any voltage			1	A
Magnetization energy consumption	approximate		4		J

Table 1.3: Power supply characteristics

1.3 Self-diagnostics and fault handling

Self-diagnostic capabilities enable the device to detect and report malfunctions. Upon detection of a problem, the device will enter a latched *fault* state and will not respond to magnetization commands until the fault is cleared, or the device is restarted.

[2] https://forum.opencyphal.org/uploads/short-url/dbQwKsDMWiMVR0Q46d1yMqR6wAv.pdf
[3] https://opencyphal.org

Fault state reporting and clearing vary by communication interface, detailed in subsequent chapters.

Failure modes detectable by the built-in diagnostics include malfunctions of the high-voltage power stage electronics, related sensor suite, and invalid configuration parameters. The latter fault mode should be addressed by correcting the configuration parameters via the Cyphal interface and then restarting the device. In addition, a deep brownout of the power supply may cause the device to fail to magnetize or demagnetize the magnet, which may also trigger a fault state.

1.4 Absolute maximum ratings

Exceeding these limits may cause permanent device damage. The maximum internal temperature is limited by the heat deflection temperature of the polymer composite body of the device, while the electronics is designed to withstand 105°C continuously.

Parameter	Min	Max	Unit
Supply voltage	-0.3	+35	V
Temperature, ambient	-40	+70	°C
Temperature, internal		+85	°C
Analog port input voltage	-0.3	+7	V
CAN FD H/L input voltage	-24	+24	V
Mechanical stress, any direction, magnitude of		250	N

Table 1.4: Absolute maximum ratings

1.5 Mechanical characteristics

The body of the device is manufactured from a mechanically and chemically resilient polymer composite. The device is watertight (operable when submerged), dust-tight, and highly resilient to vibration and adverse chemicals.

The power supply and interface connectors are installed on cables exposed via the top surface of the device. The cables are terminated with connectors documented in the respective sections.

The mechanical dimensions and the placement of the interface cables and LED indicators are shown in figure 1.1.

Parameter	Value	Unit
Mass	72	g
Ingress protection	IP69	
CAN interface cable length	12 ± 3	cm
Analog interface cable length	16 ± 4	cm

Table 1.5: Mechanical characteristics

1.6 Magnetic characteristics

The holding force is measured against a 4 mm thick polished iron sheet in full contact with the magnet poles, with the force applied against the geometric center of the magnet perpendicular to the sheet surface. Lower magnetic permeability, magnetic saturation, or imperfect contact will reduce the holding force. Force applied with an offset relative to the geometric center of the magnet will also reduce the holding force.

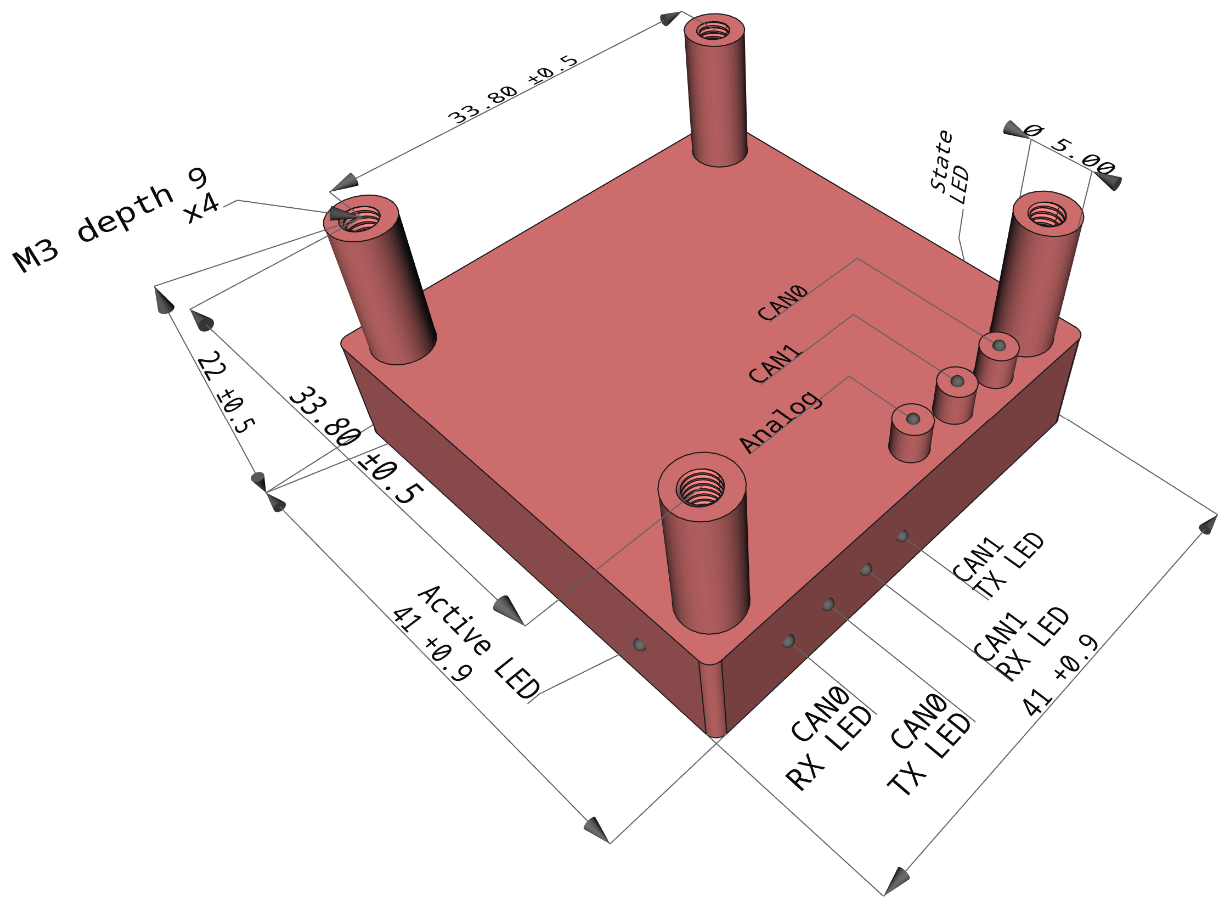
Parameter	Value	Unit
Holding force	250	N

Table 1.6: Magnetic characteristics

1.7 Reliability and safety

The device is built with AEC-Q grade 2 qualified electronic components.

For further information on reliability, quality, and functional safety, contact Zubax Robotics.



All dimensions are in millimeters. Interface cables not shown. Some interfaces are only available in specific models; refer to table 1.2 for details.

Figure 1.1: FG40 drawing

Parameter	Value	Unit
Replacement life	8	year
Operational service life	50 000	hour
Magnetization cycle limit	5000	cycles
Mean time to failure (MTTF)	*on request*	hour

Table 1.7: Reliability

1.7.1 Interference with sensitive equipment

The product has not been found to cause adverse effects on sensitive navigation equipment under typical conditions[4].

1.7.2 Quality assurance

Every manufactured device undergoes automated verification, with the records available on the web[5]. To facilitate traceability and reduce the risk of counterfeits, every manufactured device stores a unique identifier and a digital certificate of authenticity (CoA) in its non-volatile memory.

[4] A detailed report on magnetic interference is available at https://forum.zubax.com/t/2275.
[5] https://device.zubax.com/device_info

2 LED indication

The device includes several LED indicators, with their locations shown in figure 1.1.

2.1 Normal operation

2.1.1 State LED

The State LED displays different fading or blinking patterns to indicate the device's current state. The pattern is repeated every second.

LED behavior	Indicated device state
1 blink	Idle, demagnetized
2 blinks	Idle, magnetized
Fade in	Magnetizing
Fade out	Demagnetizing
Fade in-out	Idle, magnet state unknown
Continuous blinking	Fault, section 1.3

Table 2.1: State LED behaviors

2.1.2 Activity LED

The Activity LED illuminates when the device is performing nontrivial activities.

2.1.3 CAN LED

The device contains two LED indicators per CAN interface, labeled RX and TX.

While the CAN controller resides in the error-active state[6], the RX LED illuminates briefly upon successful receipt of a valid CAN frame accepted by the hardware filters; similarly, a TX LED illuminates briefly when the device successfully transmits a CAN frame with acknowledgement.

In the error-passive state, the TX LED blinks rapidly continuously, while the RX LED continues to operate normally. This choice reflects the fact that in the error-passive state the device can only receive CAN frames, but cannot transmit them.

In the bus-off state, both RX and TX indicators blink rapidly. This choice reflects the fact that in the bus-off state the device cannot transmit or receive CAN frames.

2.2 Bootloader execution

While the bootloader is being executed, which occurs briefly at power-on and after the new firmware image is downloaded (chapter 6), the LED indicators form a progress bar that is filled as the bootloader is making progress. Should the process fail (which may happen after an unsuccessful firmware update process), all LED indicators will display a marquee pattern with a single moving dot until the device is automatically restarted.

[6] Per the CAN bus specification, the *error-active* state is the normal operating state of the CAN controller. Refer to ISO 11898-1 for more information.

3 Cyphal interface

Cyphal, formerly known as UAVCAN until March 2022, is the primary communication protocol supported by this device. For historical reasons, registers related to the Cyphal stack may be named with the `uavcan.` prefix.

The product utilizes data types from the vendor-specific DSDL namespace `zubax`[7]. Human-friendly documentation for this and other DSDL namespaces can be viewed online via Nunaweb[8].

Magnet control is implemented via a dedicated command topic subscription (section 3.3.1). Status feedback is implemented by publishing feedback messages to a dedicated topic (section 3.2.2). Fault reset is done by sending an ExecuteCommand request with the appropriate opcode (section 3.4.2).

3.1 Plug-and-play node-ID allocation

By default, the local Cyphal node does not have an assigned node-ID; that is, the `uavcan.node.id` (page 17) register is set to 65535 ($FFFF_{16}$). When this is the case, the device will commence the standard plug-and-play (PnP) node-ID allocation sequence defined in the Cyphal specification[9].

Once the node-ID allocation is completed, the `uavcan.node.id` (page 17) register will be updated with the allocated node-ID value.

3.2 Publications

For the standard configuration registers pertaining to the non-fixed-port-ID topics, refer to section 5.1. Priority of the non-fixed-port-ID topics is configurable.

Topic name	Type	Priority	Period	Timeout	Description
	`uavcan.node.Heartbeat.1`	4 (nominal)	1 s	period	Section 3.2.1
	`uavcan.node.port.List`	7 (optional)	10 s	period	
	`uavcan.pnp.NodeIDAllocationData.1`	6 (slow)	random	1 s	Only during PnP allocation; section 3.1
`feedback`	`zubax.fluxgrip.Feedback.1`	4 (nominal)	1 s and on change	1 s	Section 3.2.2

Table 3.1: Cyphal publications

3.2.1 Heartbeat

The `mode` field takes the following values:

- `OPERATIONAL` unless the firmware update is in progress.
- `SOFTWARE_UPDATE` if a new firmware image is being downloaded; see chapter 6.

The `health` field takes the following values:

- `NOMINAL` if the device is operating correctly.
- `ADVISORY` if the device is operating correctly but minor issues are detected.
- `WARNING` if the device is in the fault state (section 1.3) and is unable to operate normally.

The `vendor_specific_status_code` (VSSC) field is split into two orthogonal 4-bit nibbles. The least significant nibble contains the general device status code, while the most significant nibble contains the firmware update process status code.

The interpretation of the general device status code (low VSSC nibble) is as follows:

[7] https://github.com/Zubax/zubax_dsdl
[8] http://nunaweb.opencyphal.org
[9] https://opencyphal.org/specification

Code	Device status
0	Idle, demagnetized
1	Idle, magnetized
2	Magnetizing
3	Demagnetizing
4	Idle, magnet state unknown
5	Fault: power stage malfunction, or the supply voltage is outside of the operating range (see table 1.3)
6	Fault: power stage sensor error
7	Fault: power stage malfunction
8	Fault: power stage malfunction
9	Fault: power stage sensor error
10	Fault: power stage malfunction
11	Fault: other
12	Reserved
13	Reserved
14	Reserved
15	Fault: invalid configuration

Table 3.2: Vendor-specific status code, low nibble

The interpretation of the software update status code (high VSSC nibble) is as follows:

Code	Firmware update status
0	Idle or in progress, depending on `mode`
1	Filesystem error
2	Network remote file read timed out
3	Filesystem error
4	Filesystem error
5	Network remote file read timed out
6	Firmware image file is too large

Table 3.3: Vendor-specific status code, high nibble

Definition of `uavcan.node.Heartbeat.1.0`; extent 12 bytes:

```
# Abstract node status information.
# This is the only high-level function that shall be implemented by all nodes.
#
# All Cyphal nodes that have a node-ID are required to publish this message to its fixed subject periodically.
# Nodes that do not have a node-ID (also known as "anonymous nodes") shall not publish to this subject.
#
# The default subject-ID 7509 is 1110101010101 in binary. The alternating bit pattern at the end helps transceiver
# synchronization (e.g., on CAN-based networks) and on some transports permits automatic bit rate detection.
#
# Network-wide health monitoring can be implemented by subscribing to the fixed subject.

uint16 MAX_PUBLICATION_PERIOD = 1   # [second]
# The publication period shall not exceed this limit.
# The period should not change while the node is running.

uint16 OFFLINE_TIMEOUT = 3          # [second]
# If the last message from the node was received more than this amount of time ago, it should be considered offline.

uint32 uptime                       # [second]
# The uptime seconds counter should never overflow. The counter will reach the upper limit in ~136 years,
# upon which time it should stay at 0xFFFFFFFF until the node is restarted.
# Other nodes may detect that a remote node has restarted when this value leaps backwards.

Health.1.0 health
# The abstract health status of this node.

Mode.1.0 mode
# The abstract operating mode of the publishing node.
# This field indicates the general level of readiness that can be further elaborated on a per-activity basis
# using various specialized interfaces.

uint8 vendor_specific_status_code
# Optional, vendor-specific node status code, e.g. a fault code or a status bitmask.

@assert _offset_ % 8 == {0}
@assert _offset_ == {56}  # Fits into a single-frame Classic CAN transfer (least capable transport, smallest MTU).
@extent 12 * 8
```

3.2.2 Feedback

This topic is published to at a low fixed frequency and possibly on change. Applications that do not require the extended status information can subscribe to this topic using primitive types like `zubax.primitive.boolean.Scalar.1` or `uavcan.primitive.scalar.Bit.1`.

Definition of `zubax.fluxgrip.Feedback.0.1`; extent 63 bytes:

```
# Zubax EPM status feedback message.
# This type is a structural subtype of uavcan.primitive.scalar.Bit.

bool magnetized
# True if the magnet is currently turned on.
# The value is a best guess if remagnetization is currently in progress.

int3 remagnetization_state
# -1 -- the magnet is being demagnetized (turning off).
#  0 -- the magnet is idle.
# +1 -- the magnet is being magnetized (turning on).

void4

@assert _offset_ == {8}

uint24[2] cycles_on_off
# The number of magnetization and demagnetization cycles commenced, respectively, since the device was powered on.
# The values are incremented immediately at the commencement of the cycle.

@extent 63 * 8
```

3.3 Subscriptions

Topic name	Type	TID timeout	Description
	`uavcan.pnp.NodeIDAllocationData.1`		Only during PnP allocation; section 3.1
`command`	`uavcan.primitive.scalar.Integer8.1`	0.5 s	Section 3.3.1

Table 3.4: Cyphal subscriptions

3.3.1 Command

The published values are interpreted as specified below; values that are not explicitly listed *shall not* be used. See table 1.1 for the state transition logic.

Value	Interpretation
0	OFF
1	ON
2	FORCE

Table 3.5: Command subscription handling

The command subscription can also accept boolean primitives `zubax.primitive.boolean.Scalar.1` or `uavcan.primitive.scalar.Bit.1`, since falsity is interpreted as zero, and truth as +1. This enables simplified control of the magnet through a simple boolean flag commanding the desired state.

Definition of `uavcan.primitive.scalar.Integer8.1.0`; extent 1 bytes:

```
int8 value
@sealed
```

3.4 RPC servers

Type	Description
`uavcan.node.ExecuteCommand.1`	Section 3.4.2
`uavcan.node.GetInfo.1`	Section 3.4.1
`uavcan.register.Access.1`	Section 3.4.3
`uavcan.register.List.1`	Section 3.4.3

Table 3.6: Cyphal RPC servers

3.4.1 Node info

The node name is reported as `com.zubax.fluxgrip`.

Contact Zubax Robotics for the details on the hardware version numbering and the certificate of authenticity (CoA) validation.

Definition of `uavcan.node.GetInfo.1.0`; extent 0 / 448 bytes:

```
# Full node info request.
# All of the returned information shall be static (unchanged) while the node is running.
# It is highly recommended to support this service on all nodes.

@sealed

---

Version.1.0 protocol_version
# The Cyphal protocol version implemented on this node, both major and minor.
# Not to be changed while the node is running.

Version.1.0 hardware_version
Version.1.0 software_version
# The version information shall not be changed while the node is running.
# The correct hardware version shall be reported at all times, excepting software-only nodes, in which
# case it should be set to zeros.
# If the node is equipped with a Cyphal-capable bootloader, the bootloader should report the software
# version of the installed application, if there is any; if no application is found, zeros should be reported.

uint64 software_vcs_revision_id
# A version control system (VCS) revision number or hash. Not to be changed while the node is running.
# For example, this field can be used for reporting the short git commit hash of the current
# software revision.
# Set to zero if not used.

uint8[16] unique_id
# The unique-ID (UID) is a 128-bit long sequence that is likely to be globally unique per node.
# The vendor shall ensure that the probability of a collision with any other node UID globally is negligibly low.
# UID is defined once per hardware unit and should never be changed.
# All zeros is not a valid UID.
# If the node is equipped with a Cyphal-capable bootloader, the bootloader shall use the same UID.

@assert _offset_ == {30 * 8}
# Manual serialization note: only fixed-size fields up to this point. The following fields are dynamically sized.

uint8[<=50] name
# Human-readable non-empty ASCII node name. An empty name is not permitted.
# The name shall not be changed while the node is running.
# Allowed characters are: a-z (lowercase ASCII letters) 0-9 (decimal digits) . (dot) - (dash) _ (underscore).
# Node name is a reversed Internet domain name (like Java packages), e.g. "com.manufacturer.project.product".

uint64[<=1] software_image_crc
# The value of an arbitrary hash function applied to the software image. Not to be changed while the node is running.
# This field can be used to detect whether the software or firmware running on the node is an exact
# same version as a certain specific revision. This field provides a very strong identity guarantee,
# unlike the version fields above, which can be the same for different builds of the software.
# As can be seen from its definition, this field is optional.
#
# The exact hash function and the methods of its application are implementation-defined.
# However, implementations are recommended to adhere to the following guidelines, fully or partially:
#   - The hash function should be CRC-64-WE.
#   - The hash function should be applied to the entire application image padded to 8 bytes.
#   - If the computed image CRC is stored within the software image itself, the value of
#     the hash function becomes ill-defined, because it becomes recursively dependent on itself.
#     In order to circumvent this issue, while computing or checking the CRC, its value stored
#     within the image should be zeroed out.

uint8[<=222] certificate_of_authenticity
# The certificate of authenticity (COA) of the node, 222 bytes max, optional. This field can be used for
# reporting digital signatures (e.g., RSA-1776, or ECDSA if a higher degree of cryptographic strength is desired).
# Leave empty if not used. Not to be changed while the node is running.

@assert _offset_ % 8 == {0}
@assert _offset_.max == (313 * 8)     # At most five CAN FD frames
@extent 448 * 8
```

3.4.2 Execute command

The standard command execution service is implemented. The following commands are supported; commands that are not explicitly listed return `STATUS_BAD_COMMAND`; commands whose command code is less than 32768 which are not listed here shall not be invoked; otherwise, permanent hardware damage may occur.

Command	Parameter	Description
0 (vendor-specific)		Clear the latched fault (section 1.3); do nothing if no fault present.
65530 = `STORE_PERSISTENT_STATES`		Alias for RESTART.
65532 = `FACTORY_RESET`		Reset all configuration registers to defaults.
65533 = `BEGIN_SOFTWARE_UPDATE`	Firmware image file name	Commence firmware update (chapter 6).
65535 = `RESTART`		Restart the device saving the configuration registers (more on registers in chapter 5).

Table 3.7: Cyphal command execution service

Definition of `uavcan.node.ExecuteCommand.1.3`; extent 300 / 48 bytes:

```
# Instructs the server node to execute or commence execution of a simple predefined command.
# All standard commands are optional; i.e., not guaranteed to be supported by all nodes.

uint16 command
# Standard pre-defined commands are at the top of the range (defined below).
# Vendors can define arbitrary, vendor-specific commands in the bottom part of the range (starting from zero).
# Vendor-specific commands shall not use identifiers above 32767.

uint16 COMMAND_RESTART = 65535
# Reboot the node.
# Note that some standard commands may or may not require a restart in order to take effect; e.g., factory reset.

uint16 COMMAND_POWER_OFF = 65534
# Shut down the node; further access will not be possible until the power is turned back on.

uint16 COMMAND_BEGIN_SOFTWARE_UPDATE = 65533
# Begin the software update process using uavcan.file.Read. This command makes use of the "parameter" field below.
# The parameter contains the path to the new software image file to be downloaded by the server from the client
# using the standard service uavcan.file.Read. Observe that this operation swaps the roles of the client and
# the server.
#
# Upon reception of this command, the server (updatee) will evaluate whether it is possible to begin the
# software update process. If that is deemed impossible, the command will be rejected with one of the
# error codes defined in the response section of this definition (e.g., BAD_STATE if the node is currently
# on-duty and a sudden interruption of its activities is considered unsafe, and so on).
# If an update process is already underway, the updatee should abort the process and restart with the new file,
# unless the updatee can determine that the specified file is the same file that is already being downloaded,
# in which case it is allowed to respond SUCCESS and continue the old update process.
# If there are no other conditions precluding the requested update, the updatee will return a SUCCESS and
# initiate the file transfer process by invoking the standard service uavcan.file.Read repeatedly until the file
# is transferred fully (please refer to the documentation for that data type for more information about its usage).
#
# While the software is being updated, the updatee should set its mode (the field "mode" in uavcan.node.Heartbeat)
# to MODE_SOFTWARE_UPDATE. Please refer to the documentation for uavcan.node.Heartbeat for more information.
#
# It is recognized that most systems will have to interrupt their normal services to perform the software update
# (unless some form of software hot swapping is implemented, as is the case in some high-availability systems).
#
# Microcontrollers that are requested to update their firmware may need to stop execution of their current firmware
# and start the embedded bootloader (although other approaches are possible as well). In that case,
# while the embedded bootloader is running, the mode reported via the message uavcan.node.Heartbeat should be
# MODE_SOFTWARE_UPDATE as long as the bootloader is runing, even if no update-related activities
# are currently underway. For example, if the update process failed and the bootloader cannot load the software,
# the same mode MODE_SOFTWARE_UPDATE will be reported.
# It is also recognized that in a microcontroller setting, the application that served the update request will have
# to pass the update-related metadata (such as the node-ID of the server and the firmware image file path) to
# the embedded bootloader. The tactics of that transaction lie outside of the scope of this specification.

uint16 COMMAND_FACTORY_RESET = 65532
# Return the node's configuration back to the factory default settings (may require restart).
# Due to the uncertainty whether a restart is required, generic interfaces should always force a restart.

uint16 COMMAND_EMERGENCY_STOP = 65531
# Cease activities immediately, enter a safe state until restarted.
# Further operation may no longer be possible until a restart command is executed.

uint16 COMMAND_STORE_PERSISTENT_STATES = 65530
# This command instructs the node to store the current configuration parameter values and other persistent states
# to the non-volatile storage. Nodes are allowed to manage persistent states automatically, obviating the need for
# this command by committing all such data to the non-volatile memory automatically as necessary. However, some
# nodes may lack this functionality, in which case this parameter should be used. Generic interfaces should always
# invoke this command in order to ensure that the data is stored even if the node doesn't implement automatic
# persistence management.

uint16 COMMAND_IDENTIFY = 65529
# This command instructs the node to physically identify itself in some way--e.g., by flashing a light or
# emitting a sound. The duration and the nature of the identification process is implementation-defined.
# This command can be useful for human operators to match assigned node-ID values to physical nodes during setup.

uint8[<=uavcan.file.Path.2.0.MAX_LENGTH] parameter
# A string parameter supplied to the command. The format and interpretation is command-specific.
# The standard commands do not use this field (ignore it), excepting the following:
#   - COMMAND_BEGIN_SOFTWARE_UPDATE

@extent 300 * 8

---

uint8 STATUS_SUCCESS        = 0     # Started or executed successfully
uint8 STATUS_FAILURE        = 1     # Could not start or the desired outcome could not be reached
uint8 STATUS_NOT_AUTHORIZED = 2     # Denied due to lack of authorization
uint8 STATUS_BAD_COMMAND    = 3     # The requested command is not known or not supported
uint8 STATUS_BAD_PARAMETER  = 4     # The supplied parameter cannot be used with the selected command
uint8 STATUS_BAD_STATE      = 5     # The current state of the node does not permit execution of this command
uint8 STATUS_INTERNAL_ERROR = 6     # The operation should have succeeded but an unexpected failure occurred
uint8 status
# The result of the request.

uint8[<=46] output
# Any output that could be useful that has the capability to convey detailed information.
# Users can send commands and receive specific data, like device status or measurements back in a streamlined manner.
# The standard commands should leave this field empty unless explicitly specified otherwise.

@extent 48 * 8
```

3.4.3 Register

The device implements the standard register service `uavcan.register` version 1. The data type definitions are not shown here for reasons of brevity; please consult with the official Cyphal documentation instead.

Firmware revisions prior to v1.1 do not support the register service. Therefore, in the older firmware versions, the configuration parameters cannot be changed from the default values.

Refer to chapter 5 for further details on the device registers.

3.5 Transports

3.5.1 Cyphal/CAN

The product supports the CAN FD bus interface, which may be doubly-redundant depending on the hardware configuration (section 1.1). The primary interface is labeled CAN0 and the redundant, if available, is labeled CAN1.

Firmware revisions prior to v1.3 do not support CAN FD; only Classic CAN is supported.

The value of register `uavcan.can.count` (page 17) reflects the number of CAN interfaces that are actually used in the application. If a redundant CAN interface is available but only one is used, the used interface shall be strictly CAN0, and the count register shall be set to 1, thereby disabling CAN1. If set to zero, the device will no longer be possible to contact via Cyphal/CAN.

The CAN bitrates are configured via register `uavcan.can.bitrate` (page 17), where the first value sets the arbitration phase bitrate and the second value sets the data phase bitrate. If the configured bitrate is not supported, the device will automatically revert to the default value, which is 1 Mbps for both arbitration and identification phases (i.e., BRS disabled).

The CAN MTU is set via `uavcan.can.mtu` (page 17); the only valid values are 8 and 64 bytes. Setting MTU to 8 bytes and both arbitration and data phase bitrates to the same value will disable CAN FD and select Classic CAN instead, which enables compatibility with legacy CAN networks. The Classic CAN compatibility is enabled by default to simplify deployment; the CAN FD support needs to be enabled explicitly if necessary.

The device does not support automatic CAN configuration detection. The factory-default CAN parameters are as follows: 1 Mbps for both arbitration and data phase bitrates, MTU 8 bytes, Classic CAN mode.

The device automatically recovers from the bus-off and error-passive states following the CAN bus specification.

3.5.1.1 CAN FD interface characteristics

The CAN FD interfaces may be equipped with different connector types upon request.

Parameter	Condition	Min	Typ	Max	Unit
Differential output voltage, dominant	60Ω load	+1.5		+5.1	V
Differential output voltage, recessive	60Ω load	-0.1	0	+0.1	V
Differential receiver threshold voltage		0.4		1.15	V
Differential receiver hysteresis voltage		0.05		0.3	V
Arbitration phase bit rate		10		1000	kbps
Data phase bit rate		10		4000	kbps

Table 3.8: CAN FD interface characteristics

3.5.1.2 UCANPHY Micro connector

The UCANPHY Micro connector is specified in the UCANPHY Specification. This connector type is based on JST GH and is crimped on a cable; an external T-connector or a hub/splitter may be necessary to attach the device to the CAN bus[10].

[10] Suitable CAN bus hubs/splitters are available from Zubax Robotics.

Pin no.	Type	Function
1	Power	4-30 V power supply input, see table 1.3
2	Input/output	CAN high
3	Input/output	CAN low
4	Ground	Ground

Table 3.9: UCANPHY Micro connector pinout

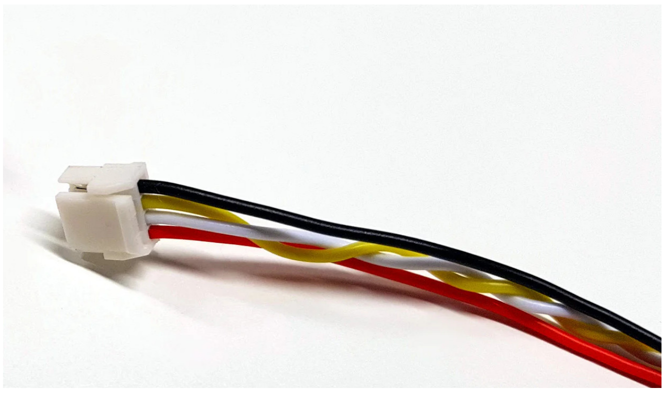


Figure 3.1: CAN bus interface cable with a UCANPHY Micro connector

4 Analog interface

The analog interface enables device control through a single pin, supporting either simple DC voltage level or the industry-standard RC PWM interface[11] as alternatives to the Cyphal interface.

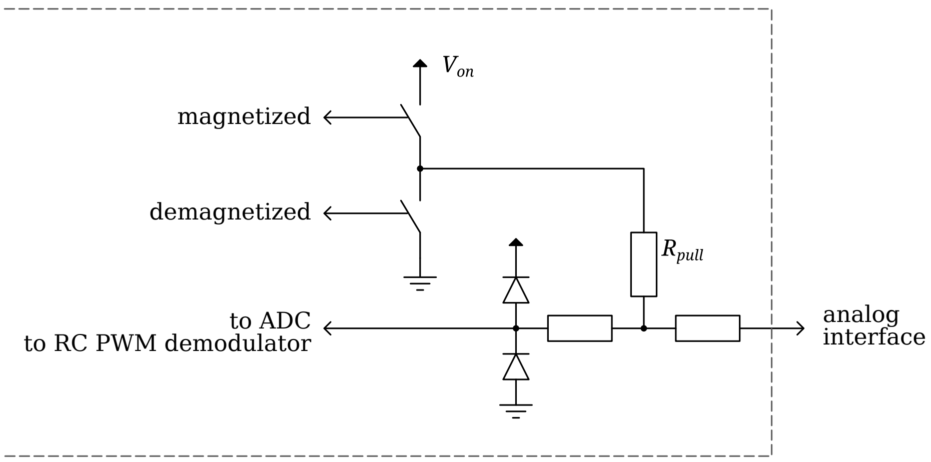


Figure 4.1: Analog interface internal schematic

Magnet state feedback is provided on the same pin by setting the internal pull resistor R_{pull} to $V_{\text{on}} = 3.3V$ when the device is magnetized and grounding it when demagnetized. If the magnet state is unknown, the pull resistor is set to $\frac{V_{\text{on}}}{2} = 1.65V$.

The device automatically detects the type of the connected interface and adjusts its behavior accordingly. This is done by checking for the presence of the RC PWM carrier signal; if such is not found, the device assumes that the voltage level control is used. If used concurrently with Cyphal, the latter takes precedence and the analog input is ignored.

To clear the latched fault state (section 1.3) via the analog interface, send the FORCE command and hold it for at least 3 seconds; then send either ON or OFF command, which will cause the fault state to be cleared and the magnet to be magnetized or demagnetized, respectively, unless another fault is triggered in the process.

4.1 RC PWM control

In RC PWM control mode, an external RC PWM signal source is connected to the analog interface input. The impedance of the source should be low enough to override the pull resistor feedback.

The RC PWM signal is sampled by the device and the magnet is commanded to the corresponding state as specified in table 4.1. See table 1.1 for the state transition logic.

Min	Max	Interpretation
0.8 ms	1.2 ms	OFF
1.6 ms	1.9 ms	ON
2.1 ms	2.5 ms	FORCE

Table 4.1: RC PWM pulse duration thresholds

Parameter	Min	Max	Unit
Pulse frequency	40	400	Hz
Low level duration	0.1		ms

Table 4.2: RC PWM characteristics

[11] https://en.wikipedia.org/wiki/Servo_control

4.2 Voltage level control

In the voltage level control mode, a low-impedance external voltage source is connected to the analog interface input such that it overrides the pull resistor feedback.

The device measures the voltage at the analog interface, V_{analog}, to command the magnet to the corresponding state as detailed in table 4.3. See table 1.1 for the state transition diagram.

A debouncer is applied to the input signal to prevent spurious transitions.

Min	Max	Interpretation
0.0V	0.8V	OFF
2.5V	3.8V	ON
4.4V	6.0V	FORCE

Table 4.3: Voltage level thresholds

The pull resistor feedback ensures that the magnet will reside in the current state if the input is left floating.

4.2.1 Typical application examples

4.2.1.1 3.3 V logic level control

An ordinary GPIO pin of a microcontroller can be used to control the magnet via the analog interface. If the application does not require the use of the FORCE command (which can be substituted with cycling the magnet off and on), the GPIO output can be connected directly to the analog interface input.

4.2.1.2 5 V logic level control

A 5 V logic level GPIO can be interfaced with the magnet in two ways: either through a voltage divider or by using an input-output pin.

The voltage divider will reduce the logic high level to the 3.3 V nominal, effectively rendering the circuit equivalent to the 3.3 V logic level control example above. This approach cannot leverage the FORCE command.

The input-output pin approach enables three possible states of the output:

- **High impedance** (i.e., input): the current magnet state is maintained unchanged. This is ensured by the pull resistor feedback. The current state of the magnet can be read by sampling the voltage level at the pin.
- **Low level**: the OFF command is issued.
- **High level**: the FORCE command is issued. After the FORCE command is issued, the output should be switched to the high impedance state (input) to allow the pull resistor feedback to maintain the magnet state.

4.2.1.3 Push button control

External push buttons can be introduced for idempotent manual control of the magnet, as illustrated in the schematic. More sophisticated circuits enabling concurrent use of the ON and FORCE commands can be devised based on these basic examples.

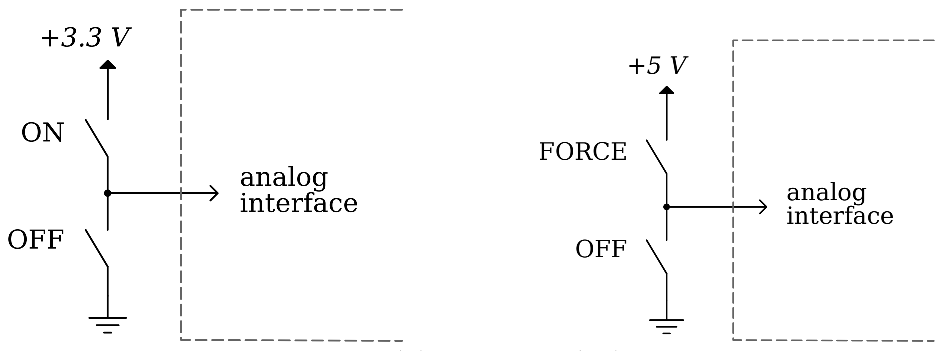


Figure 4.2: Push button control schematics

4.3 Characteristics

The analog interface is implemented on a wire terminated with the industry-standard 2.54 mm three-pin RC PWM[12] connector. Additional connector options are available upon request.

Parameter	Min	Typ	Max	Unit
Logic input low			1.0	V
Logic input high	3.8			V
Pull resistance	4800	5100	6000	Ω

Table 4.4: Analog interface characteristics

Pin no.	Type	Function
1	Input	Analog input with pull resistor feedback
2	Power	Power supply input, see table 1.3
3	Ground	Ground

Table 4.5: Analog (RC PWM) connector pinout

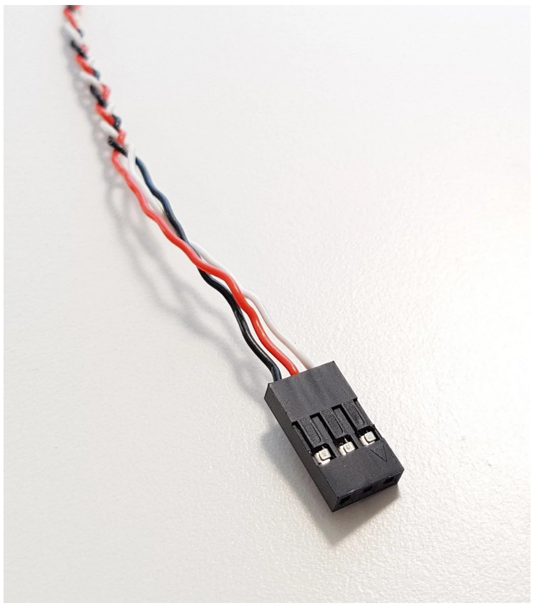


Figure 4.3: Analog interface cable with the industry-standard RC PWM connector

[12]https://en.wikipedia.org/wiki/Servo_control

5 Registers

Configuration parameters and diagnostic information are accessible via Cyphal registers. According to the Cyphal specification, four types of registers are defined: mutable/immutable × persistent/volatile. Configuration parameters are exposed as mutable persistent registers.

Changes to configuration parameters may take effect immediately or only after a device restart, as specified in each parameter's documentation. By default, one should assume the latter unless documented otherwise.

Configuration parameters (i.e., mutable persistent registers) are not committed to the non-volatile memory until the device is commanded to restart.

This document purposefully omits the description of some registers. These are not intended for production use and should not be relied upon; their name, type, contents, and semantics may change arbitrarily between minor releases with no regard for compatibility.

5.1 Configuration parameter index

Name	Type	Unit	Default	Pages	Description
`uavcan.node.id`	`natural16`		65535	6	Node ID of the local Cyphal node; 65535 enables PnP
`uavcan.node.description`	`string`		“” (empty)		User-defined string descriptor for node identification
`uavcan.can.mtu`	`natural8`	byte	8	12	Cyphal/CAN maximum transmission unit, either 8 or 64
`uavcan.can.bitrate`	`natural16[2]`	bps	1000000, 1000000	12	Arbitration and data phase bitrates
`uavcan.can.count`	`natural8`			12	Number of Cyphal/CAN interfaces to use
`uavcan.pub.feedback.id`	`natural16`		65535		`feedback` publication topic-ID
`uavcan.pub.feedback.prio`	`natural8`		4		`feedback` publication priority
`uavcan.sub.command.id`	`natural16`		65535		`command` subscription topic-ID

Table 5.1: Configuration parameter index

6 Embedded bootloader

The device incorporates an embedded bootloader that allows the firmware to be updated in the field via the Cyphal interface (chapter 3). The bootloader verifies firmware integrity at each boot.

While the bootloader is running, a special LED pattern is displayed as documented in section 2.2.

The firmware update sequence is as follows:

1. A firmware update request is sent to the device; see section 3.4.2.
2. The device initiates asynchronous firmware image download process over the Cyphal network. While the download is in progress, the device remains operational and continues to respond to commands. The bootloader is not yet running at this stage. This stage may take several minutes to complete.
3. After a successful download, the device automatically restarts to apply the new firmware. If the download fails, a status code indicating the failure reason is indicated via the Heartbeat message (section 3.2.1).
4. Upon restart, the device may require up to a few minutes to verify the downloaded file and replace the installed firmware. Should the downloaded file fail verification, the device will revert to the previously installed firmware version.
5. The node info service can be used to check whether the last firmware update was successful; see section 3.4.1.