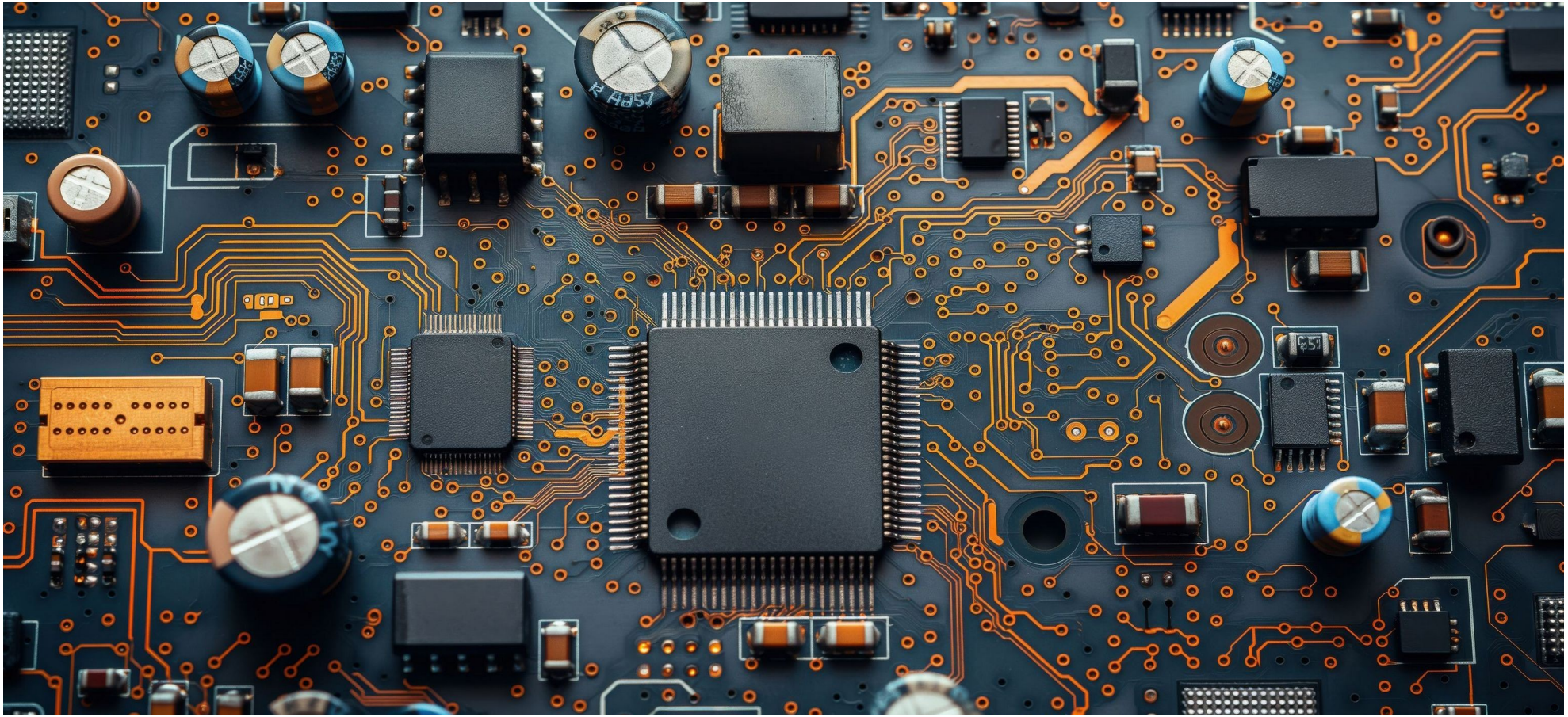
Module Development

LIBRE CUBE
open source space exploration

Release date: 2025-05-27



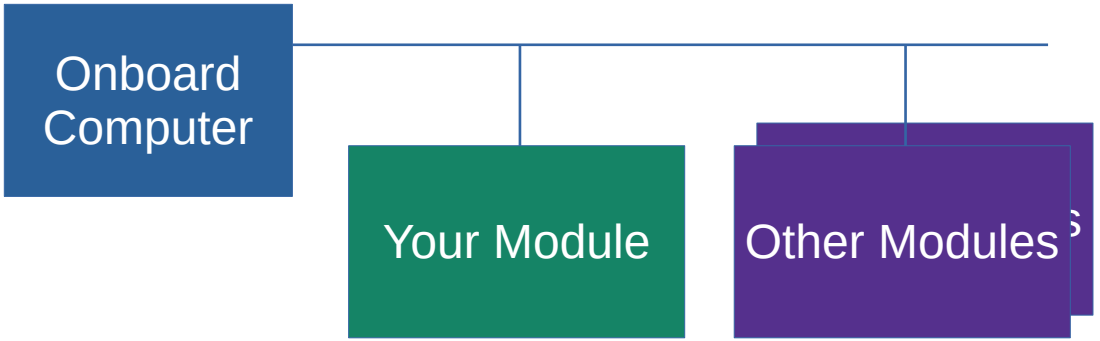
Overview

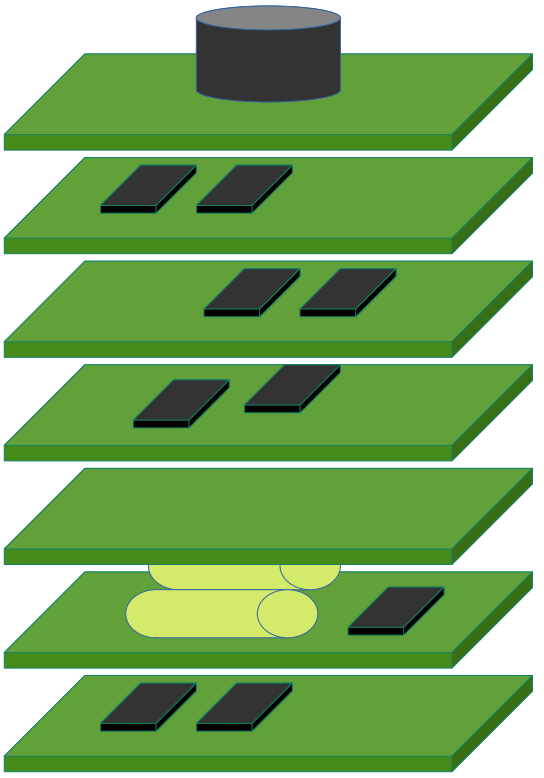
- This tutorial shows you how to develop a module that can be integrated into a system, such as a rover, satellite, drone, etc.
- You define the functionality of the module.
  - It can be a full-blown subsystem (ie. provide electrical power, propulsion, navigation) or any kind of sensor or actuator relevant for the mission.
- The module shall conform to the LibreCube board specification.

Module Stack

LIBRE CUBE
open source space exploration

- The system (rover, etc.) will consist of a stack of modules (electronic boards) that in total provide all the functionality for the mission.
- Typically a central onboard computer monitors and controls all the modules in the stack.



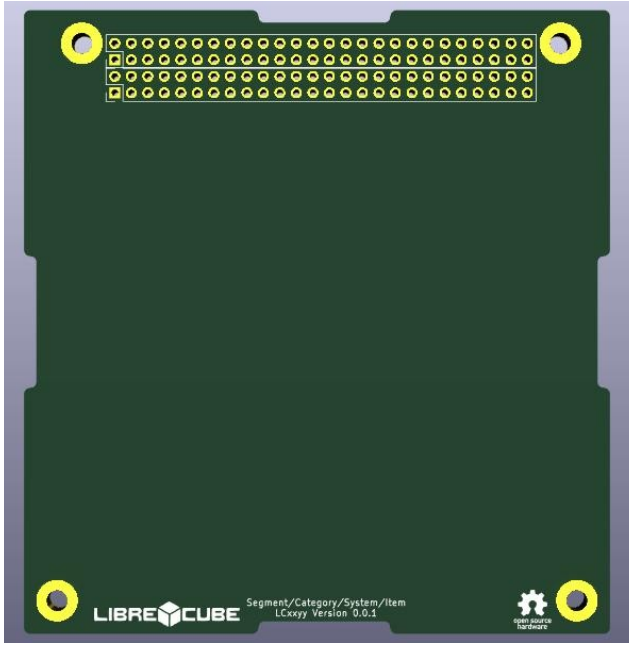


Interfaces

Your module needs to adhere to three types of interfaces:

- **Mechanical**: the layout of the board.
- **Electrical**: the electrical connections of the header, which supplies power and signals.
- **Software**: The SpaceCAN bus is used to control and monitor your module. It is defined by the SpaceCAN specification. (For payload modules, the UART bus may be used as well).

https://librecube.gitlab.io/standards/board_specification/



Development Workflow

Development Workflow

LIBRE CUBE
open source space exploration

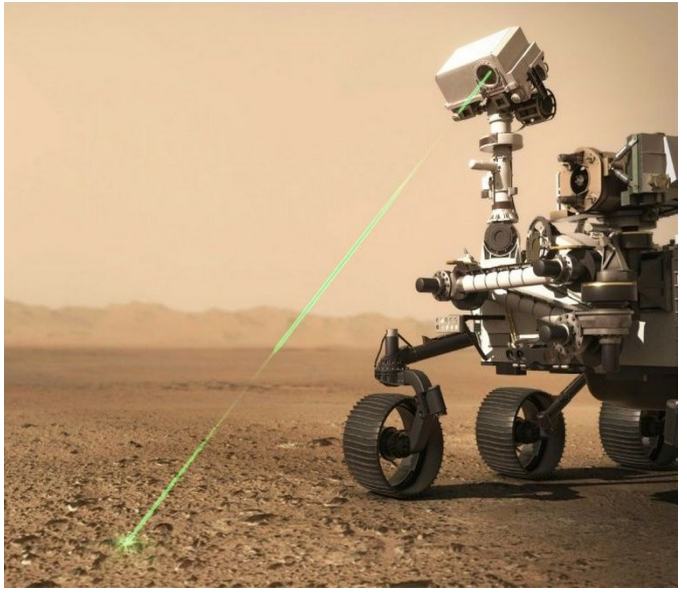
The typical workflow when developing a module from scratch is:

1. Define the functionality
2. Make design and obtain the hardware
3. Make a breadboard setup and do development
4. Prototype the SpaceCAN interface
5. Integration
6. Testing
7. Production

1. Define the Functionality

LIBRECUBE
open source space exploration

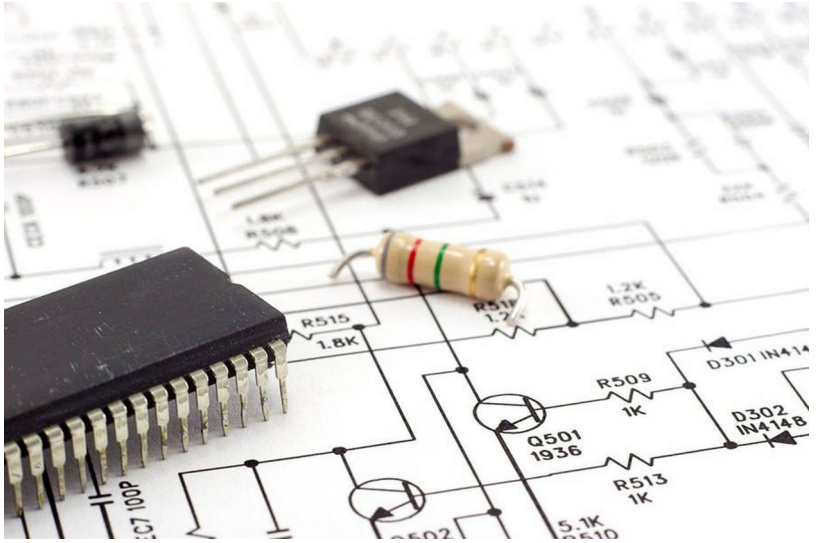
- You have full freedom in defining the functionality of your module, as long as it fits the (mission defined) overall system budget in terms of mass, volume, power, costs, etc.
- Ideas for modules:
  - GNSS receiver module
  - Camera module
  - Lidar module
  - Laser module
  - Robotic arm module
  - LCD display module
  - Gamma spectrometer module



2. Design and Hardware

LIBRECUBE
open source space exploration

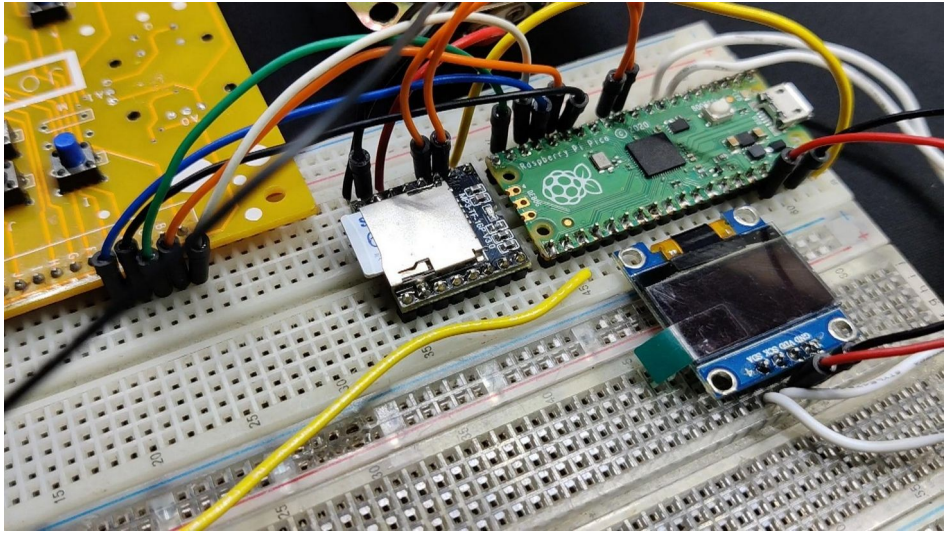
- Typically, the core functionality is achieved with a central component (for example a camera chip), which needs additional peripheral components (resistors, capacitors, etc.)
- Also, you will need a microcontroller and some circuitry for CAN bus interface, to later make it work with SpaceCAN bus.
- For power supply you can use 5V directly, or add DC/DC converters as needed.
- Sketch out the schematic and buy the parts.



3. Breadboard Development

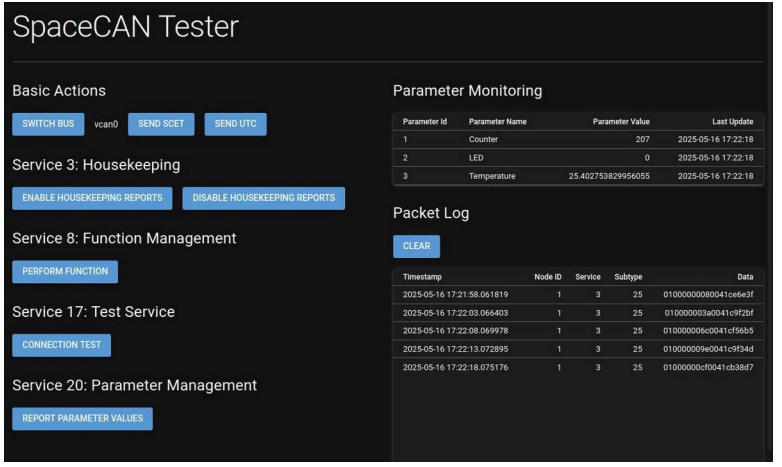
LIBRE CUBE
open source space exploration

- With the breadboard setup you verify that your schematic design is working correctly.
- The main focus is on testing the intended functionality of your module.
- You should also verify that the dual CAN interface is working. For this, send some data over both CAN busses.



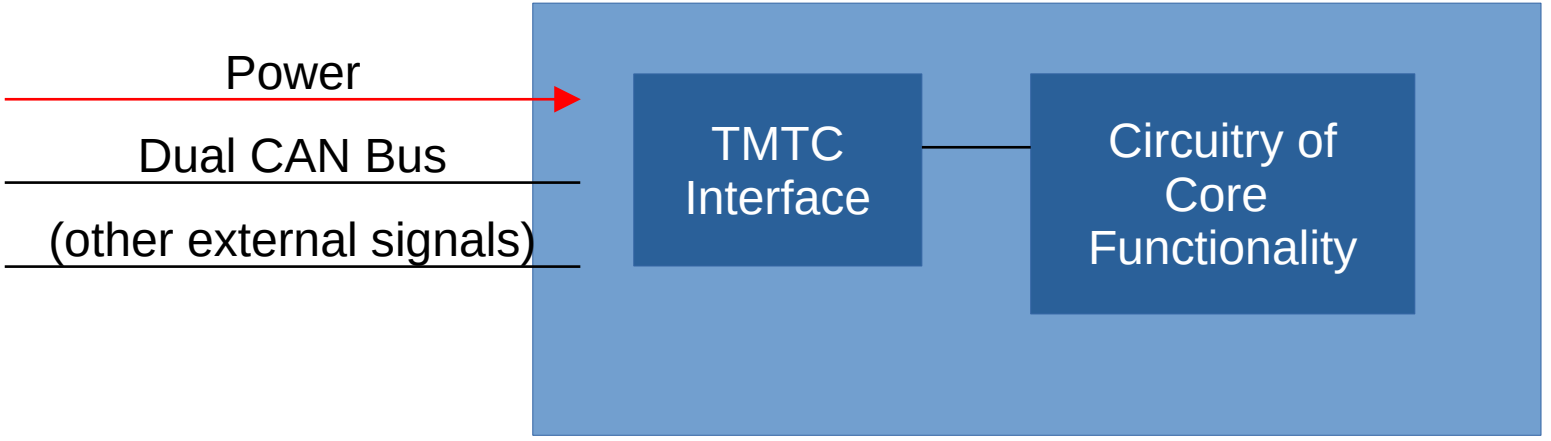
4. Prototype the SpaceCAN Interface

- Your module will be monitored and controlled by the onboard computer through the SpaceCAN interface.
- You can already prototype this interface at this point. This can be a two step approach:
  - First you can create a fully virtual setup using the Python SpaceCAN Tester and writing the config files and responder dummy code.
  - Then you can move this to a physical setup, where you use a USB to CAN adapter and your CAN node.
- In either case, consult the SpaceCAN documentation and tutorial.



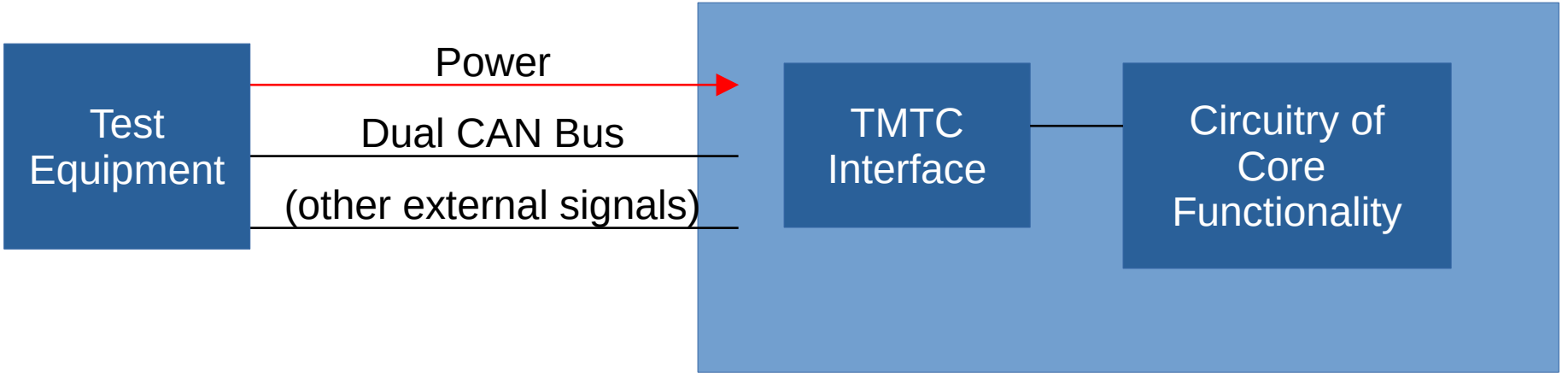
5. Integration

- Now it is time to bring the two pieces together:
  - the electronics of your module functionality (ie. the breadboard)
  - the SpaceCAN interface, also called TMTC interface



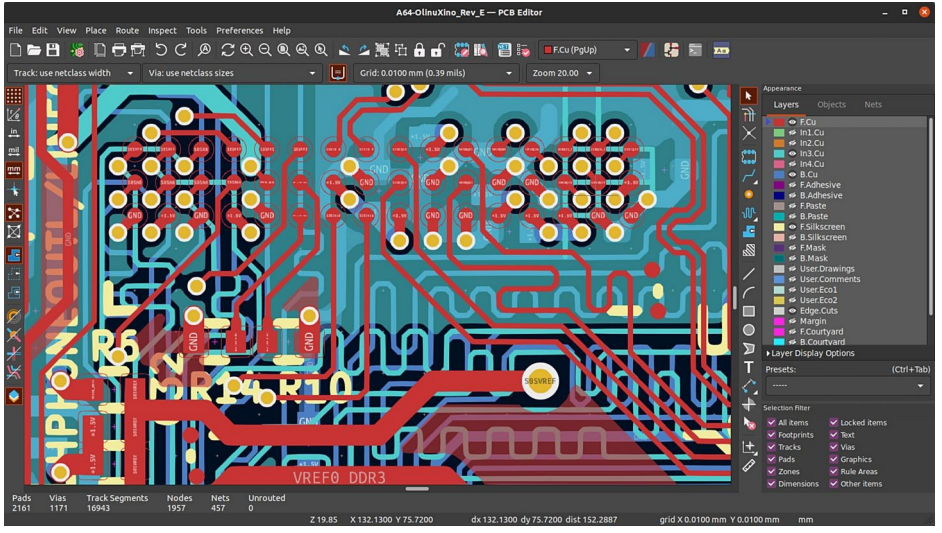
6. Testing

- With the help of test equipment (or a full-blown onboard computer) you can test now all aspects of monitoring and controlling your module.



7. Production

- At the very end, you may turn your working module into a production ready electronics board.
- You may use the LibreCube board template as starting point, add your components, do the wiring and routing, and give it to production.
- All this can be done with KiCAD.
- Finally, do the soldering and program the firmware into the device.



Practical Example: GPS Receiver Module

1. Define the Functionality

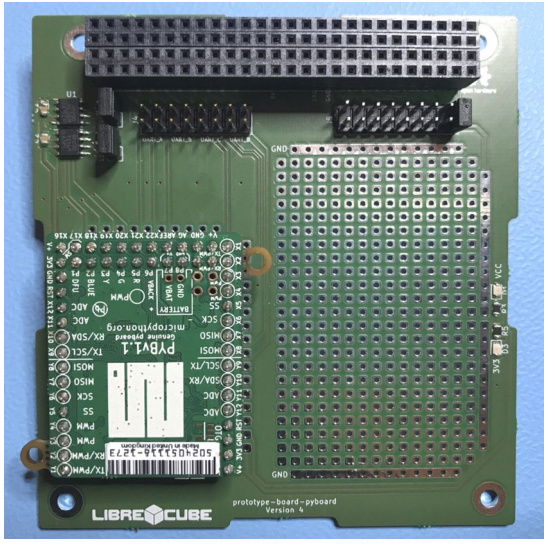
LIBRECUBE
open source space exploration

- For this practical example we want to use a low cost, low power, commercially available GPS receiver.
- The data to be provided from the module (telemetry) shall be:
  - Status of GPS mode (normal or standby)
  - Status of data (valid/invalid)
  - Current UTC time and data
  - Current position (lat, lon)
- We want to command the module as following:
  - Set GPS receiver to normal or standby mode
  - Trigger warm restart (reset configuration but use ephemeris)
  - Trigger cold restart (full reset)

2. Design and Hardware (1/4)

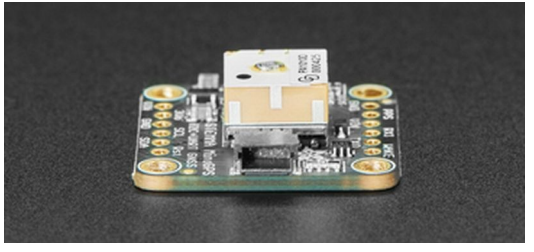
- We will use the “LibreCube Pyboard Prototype Module” because it provides the dual SpaceCAN interface already, so we can focus on the development of the GPS integration.

  https://gitlab.com/librecube/elements/LC3102



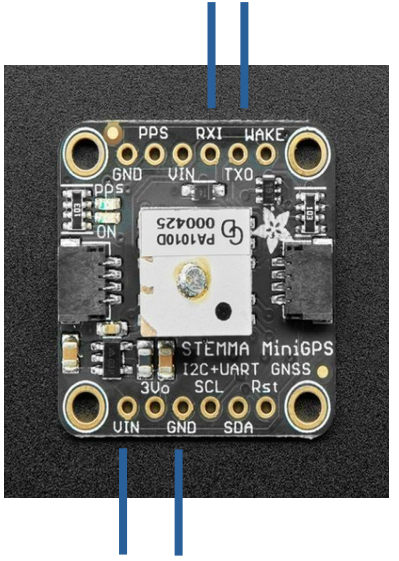
- For the GPS receiver unit we choose the Adafruit Mini GPS PA1010D.

  https://learn.adafruit.com/adafruit-mini-gps-pa1010d-module/overview



2. Design and Hardware (2/4)

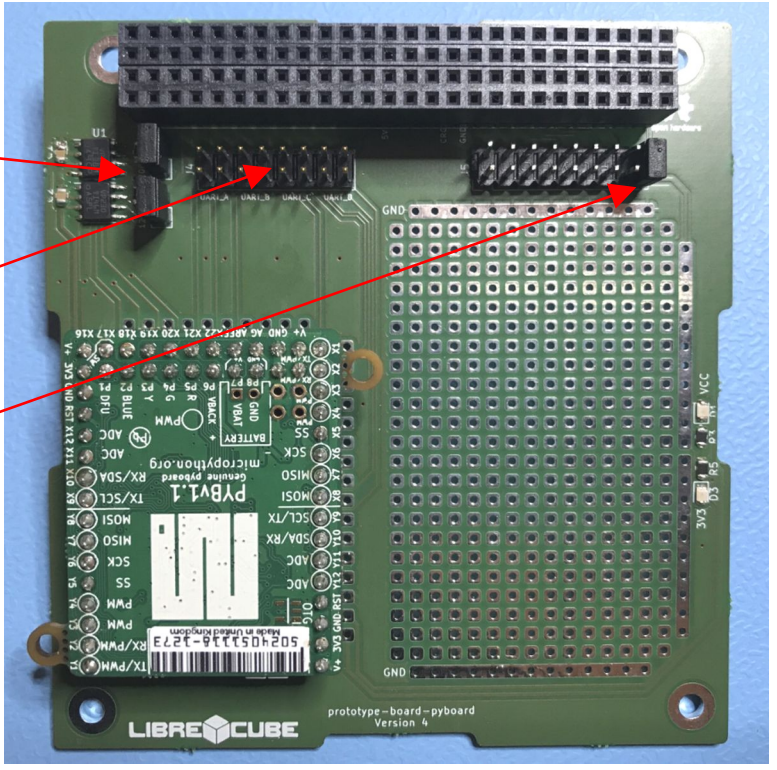
- According to the datasheet of the GPS unit [1] we will need to connect the following pins:
  - GND
  - VIN: 3.3 Volt input
  - TXO: UART TX out
  - RXI: UART RX in



[1] https://cdn-learn.adafruit.com/assets/assets/000/084/295/original/CD_PA1010D_Datasheet_v.03.pdf?1573833002

2. Design and Hardware (3/4)

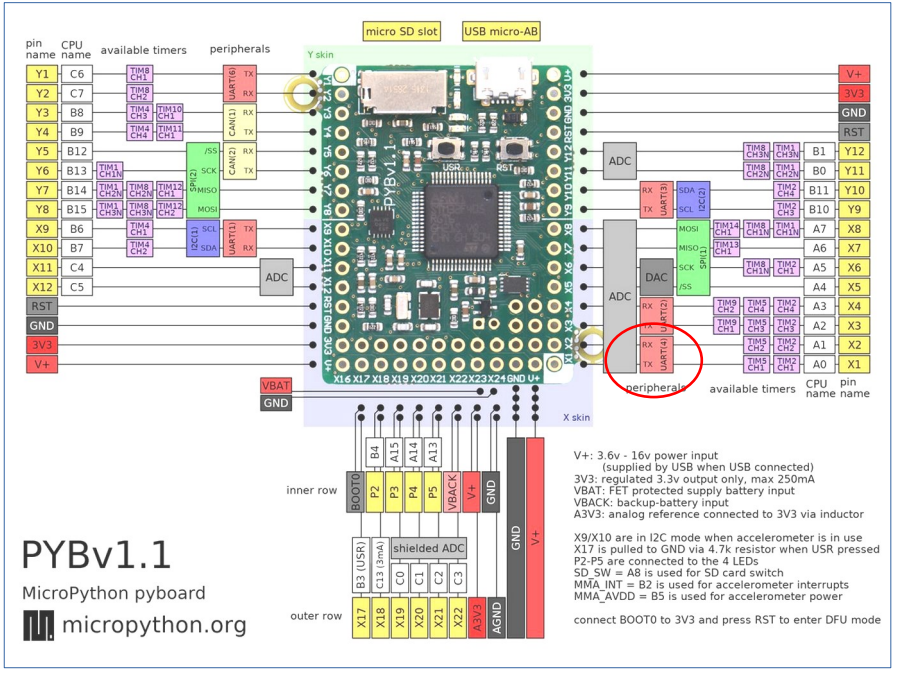
- For the Pyboard Prototype Module:
  1. Set the two 120 Ohm resistor jumpers for CAN bus.
  2. Remove all the jumpers from UART header (as we do not use the system bus UART).
  3. Select the permanent 5V line as power source.



2. Design and Hardware (4/4)

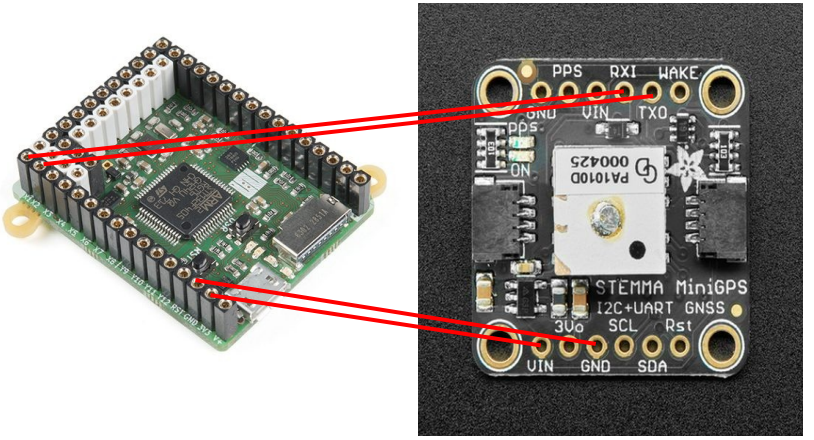
LIBRECUBE
open source space exploration

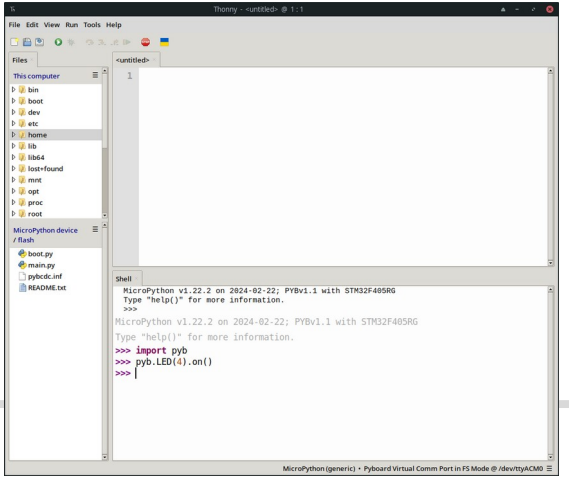
- The VIN and GND for the GPS unit will be supplied by the 3V3 and GND bar on the Pyboard Prototype Module.
- For the UART connection (TX and RX pins) between GPS unit and Pyboard we can use for example the UART 4 of pyboard.
- Note that connection must be:
  - RX(GPS)-TX(PYB) and
  - TX(GPS)-RX(PYB).



3. Breadboard Development (1/3)

- To start the development, we can simply use the pyboard and connect the UART TX & RX and the 3V3 & GND lines from it to the GPS unit.
- Connect the laptop via USB cable to the pyboard and use the Thonny IDE (https://thonny.org/) to get to the pyboard prompt.
- For a start, let’s switch on an LED of the pyboard.

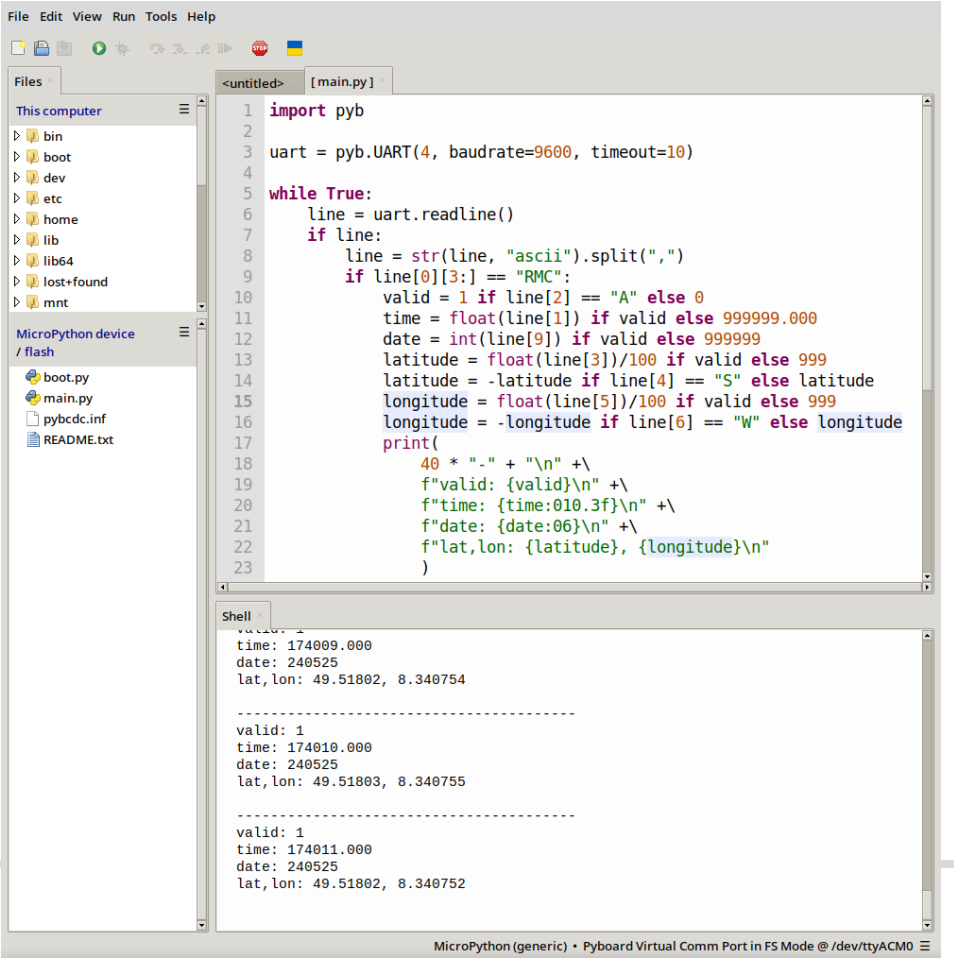




3. Breadboard Development (2/3)

LIBRECUBE
open source space exploration

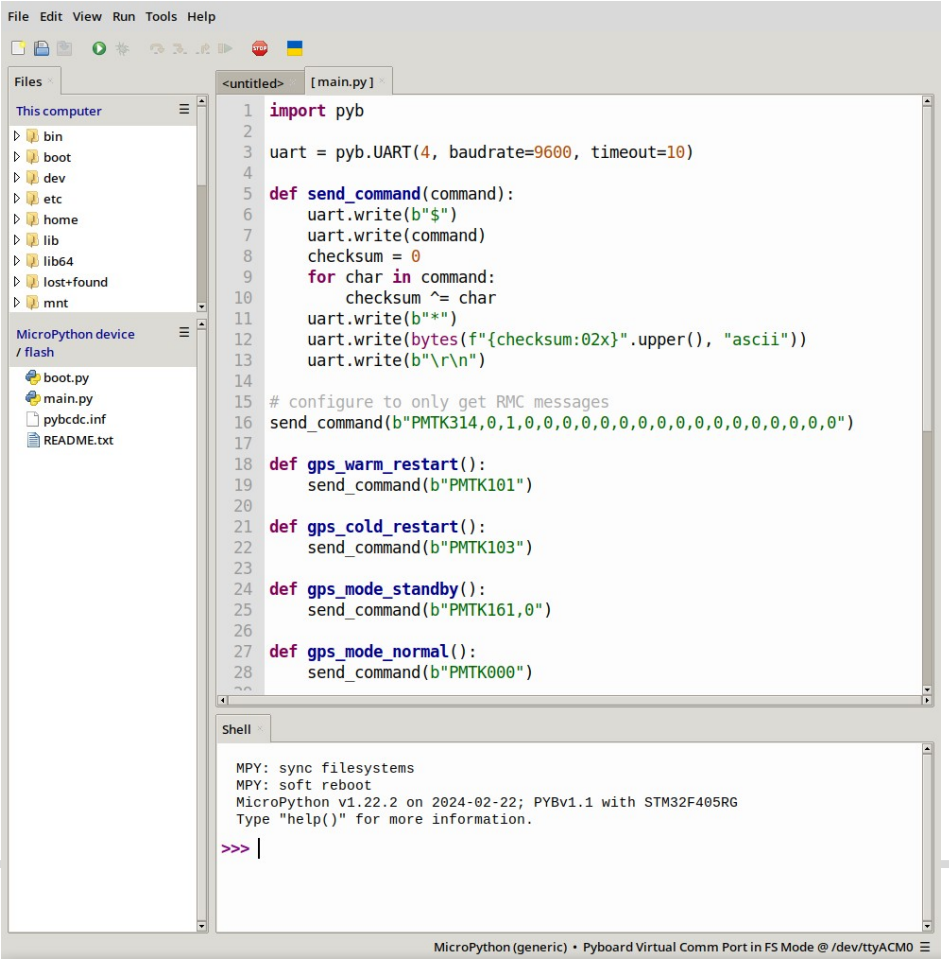
- Next step is to read and parse the messages from GPS unit.
- Consult the GPS receiver manual to learn how to parse the NMEA messages that come from the GPS.



3. Breadboard Development (3/3)

LIBRE CUBE
open source space exploration

- Finally we write some functions to control the GPS mode.
- We can see for example the proper function of the cold restart as the GPS unit will take some time to re-acquire the fix (and blink the LED).

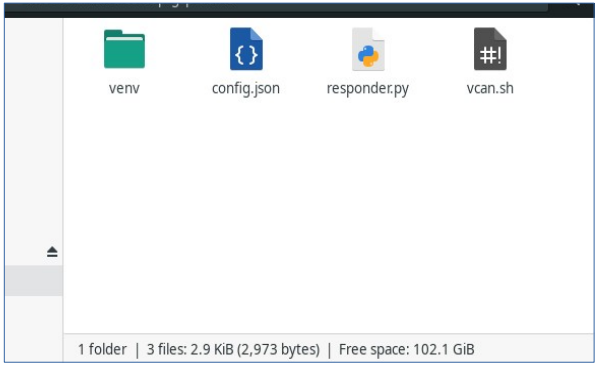


4. Prototype SpaceCAN Interface (1/4)

LIBRECUBE
open source space exploration

- For this step we only need a laptop, no other hardware!
- Independent to the development of the software to interface with the GPS unit, we can already work on the code parts needed to accommodate the SpaceCAN interface.
- Create a folder “gps-demo” and copy the files “config.json”, “vcan.sh” and “responder.py” from the SpaceCAN Tester example to it.
  https://gitlab.com/librecube/elements/LC3301/-/tree/main/examples/virtual
- Inside the folder, create the Python virtual environment and install everything:

```
python -m venv venv
source venv/bin/activate
pip install git+https://gitlab.com/librecube/elements/LC3301.git
```



4. Prototype SpaceCAN Interface (2/4)

LIBRECUBE
open source space exploration

- Modify the “config.json”:
  - Define 6 parameters to capture the information we require.
  - Define a single housekeeping report, which contains all the parameters.
  - Further, define one function with an argument to set the mode, and another two functions without arguments to trigger the restarts.

```
"use_packets": true,

"parameters": [
  {
    "parameter_id": 1,
    "parameter_name": "mode",
    "encoding": "B",
    "value": 1
  },
  {
    "parameter_id": 2,
    "parameter_name": "data valid",
    "encoding": "B",
    "value": 0
  },
  {
    "parameter_id": 3,
    "parameter_name": "utc time",
    "encoding": "L",
    "value": 999999
  },
  {
    "parameter_id": 4,
    "parameter_name": "utc date",
    "encoding": "L",
    "value": 999999
  },
  {
    "parameter_id": 5,
    "parameter_name": "latitude",
    "encoding": "f",
    "value": 999.0
  },
  {
    "parameter_id": 6,
    "parameter_name": "longitude",
    "encoding": "f",
    "value": 999.0
  }
],
```

```
"housekeeping_reports": [
  {
    "report_id": 1,
    "comment": "reports all parameters",
    "interval": 5,
    "enabled": true,
    "parameter_ids": [1, 2, 3, 4, 5, 6]
  }
],

"functions": [
  {
    "function_id": 1,
    "function_name": "Set GPS mode",
    "arguments": [
      {
        "argument_id": 1,
        "argument_name": "mode",
        "encoding": "B"
      }
    ]
  },
  {
    "function_id": 2,
    "function_name": "trigger GPS warm restart"
  },
  {
    "function_id": 3,
    "function_name": "trigger GPS cold restart"
  }
]
```

4. Prototype SpaceCAN Interface (3/4)

LIBRECUBE
open source space exploration

- Modify the “responder.py” file to implement dummy values for the parameters and dummy actions for the functions.

File Edit Search View Document Help

```python
from datetime import datetime
import time
import random
import spacecan

responder = spacecan.Responder.from_file("config.json")
services = spacecan.Services(responder).from_file("config.json")
set_param = services.parameter.set_parameter_value
get_param = services.parameter.get_parameter_value

PARAM_DATA_VALID = 1
PARAM_MODE = 2
PARAM_UTC_TIME = 3
PARAM_UTC_DATE = 4
PARAM_LAT = 5
PARAM_LON = 6

FUNC_SET_MODE = 1
FUNC_SET_MODE_ARG_MODE = 1
FUNC_WARM_RESTART = 2
FUNC_COLD_RESTART = 3


def perform_function(function_id, arguments):
    if function_id == FUNC_SET_MODE:
        print("Execute: set mode")
        if arguments[FUNC_SET_MODE_ARG_MODE] == 0:
            set_param(PARAM_MODE, 0)
            set_param(PARAM_DATA_VALID, 0)
        elif arguments[FUNC_SET_MODE_ARG_MODE] == 1:
            set_param(PARAM_MODE, 1)
            set_param(PARAM_DATA_VALID, 1)
    elif function_id == FUNC_WARM_RESTART:
        print("Execute: warm restart")
        set_param(PARAM_MODE, 1)
    elif function_id == FUNC_COLD_RESTART:
        print("Execute: cold restart")
        set_param(PARAM_MODE, 1)

```

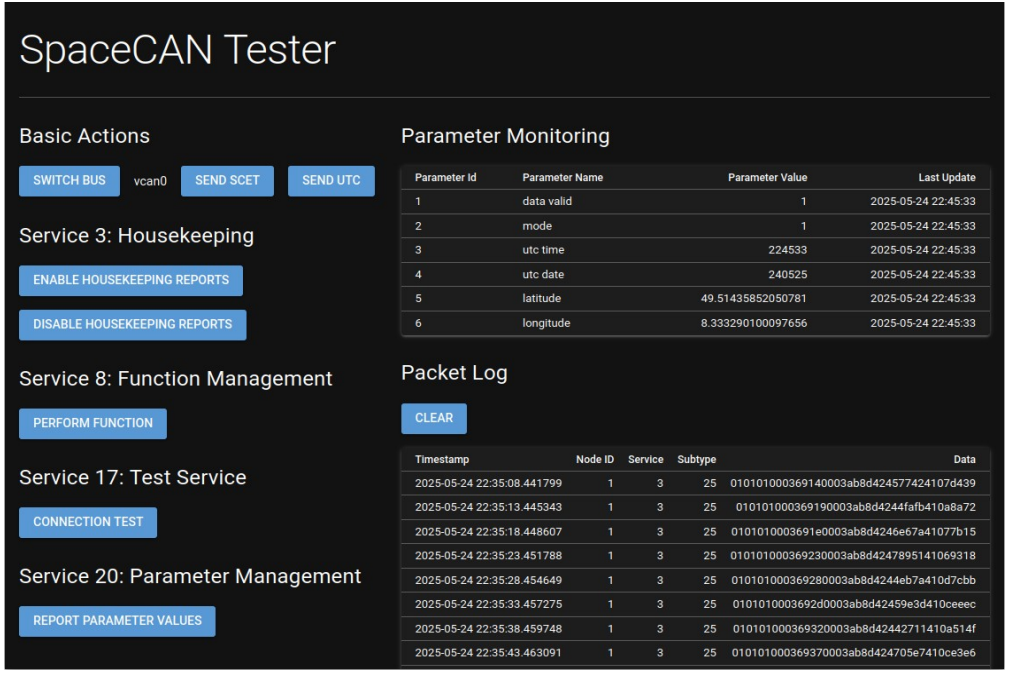
```python
    elif function_id == FUNC_COLD_RESTART:
        print("Execute: cold restart")
        set_param(PARAM_MODE, 1)


services.function.perform_function = perform_function
responder.connect()
responder.start()

try:
    set_param(PARAM_DATA_VALID, 1)
    while True:
        # do some other stuff
        time.sleep(0.1)

        # update housekeeping data
        set_param(PARAM_UTC_TIME, int(datetime.now().strftime("%H%M%S")))
        set_param(PARAM_UTC_DATE, int(datetime.now().strftime("%d%m%y")))
        set_param(PARAM_LAT, random.uniform(49, 50))
        set_param(PARAM_LON, random.uniform(8, 9))

except KeyboardInterrupt:
    pass

responder.stop()
responder.disconnect()
print("Stopped")

```

Filetype: Py

4. Prototype SpaceCAN Interface (4/4)

LIBRECUBE
open source space exploration

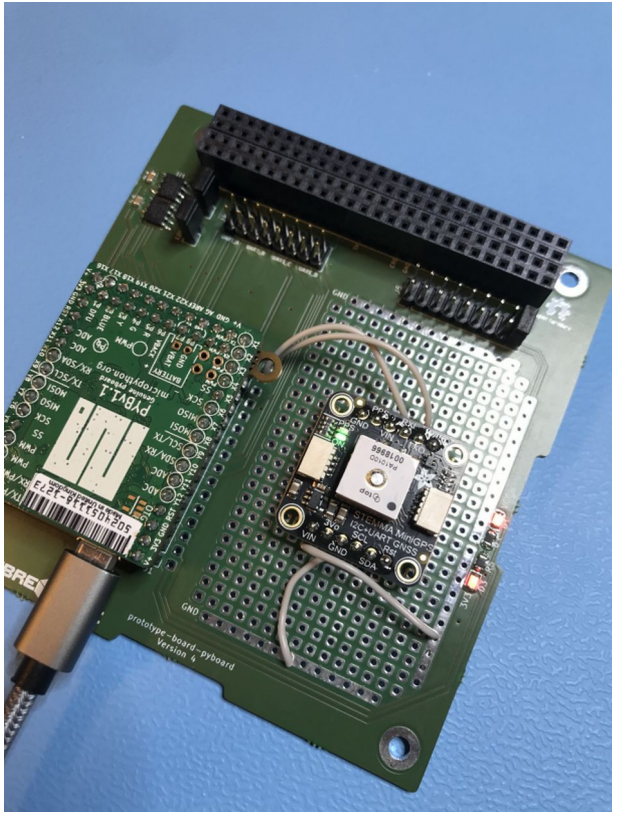
- Test the correct implementation:
  - Bring up the virtual CAN: `source vcan.sh`
  - Start the SpaceCAN Tester: `spacecan-tester config.json`
  - Start the responder: `python responder.py`



5. Integration (1/3)

LIBRE CUBE
open source space exploration

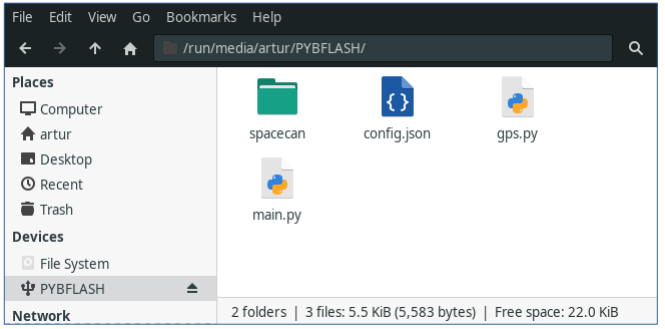
- In this step now we integrate now both the hardware parts and the software.
- For the hardware, the GPS unit can be mounted on the prototype module.
- For the software, we need to merge the code for the SpaceCAN interface and the one for handling the GPS. See next slides.



5. Integration (2/3)

LIBRECUBE
open source space exploration

- Copy to the pyboard the spacecan folder from the micropython-spacecan repository: https://gitlab.com/librecube/lib/micropython-spacecan/-/tree/main/src?ref_type=heads
- Also copy the “config.json” file to the pyboard. Modify its interface and channel configuration as shown.
- We will create a “main.py” for the main loop and the SpaceCAN interface, and refactor the GPS specific functionality into a class in a “gps.py” file. See next slide.




```
{
  "interface": "pyboard",
  "channel_a": 1,
  "channel_b": 2,
  "node_id": 1,
  "heartbeat_period": 0.5,
  "max_miss_heartbeat": 5,
  "use_packets": true,

  "parameters": [
    {
      "parameter_id": 1,
      "parameter_name": "data valid",
```

5. Integration (3/3)

- Note how the GPS specific functionality is refactored into the “gps.py” file (right)
- Whereas in the “main.py” file (left) we mainly have the code to provide the SpaceCAN interface.

```python
import time
import pyb
import spacecan
from gps import GpsUnit

uart = pyb.UART(4, baudrate=9600, timeout=10)
gps = GpsUnit(uart)
gps.init()

responder = spacecan.Responder.from_file("config.json")
services = spacecan.Services(responder).from_file("config.json")
set_param = services.parameter.set_parameter_value
get_param = services.parameter.get_parameter_value

PARAM_MODE = 1
PARAM_DATA_VALID = 2
PARAM_UTC_TIME = 3
PARAM_UTC_DATE = 4
PARAM_LAT = 5
PARAM_LON = 6

FUNC_SET_MODE = 1
FUNC_SET_MODE_ARG_MODE = 1
FUNC_WARM_RESTART = 2
FUNC_COLD_RESTART = 3

def perform_function(function_id, arguments):
    if function_id == FUNC_SET_MODE:
        if arguments[FUNC_SET_MODE_ARG_MODE] == 0:
            gps.mode_standby()
        elif arguments[FUNC_SET_MODE_ARG_MODE] == 1:
            gps.mode_normal()
    elif function_id == FUNC_WARM_RESTART:
        gps.warm_restart()
    elif function_id == FUNC_COLD_RESTART:
        gps.cold_restart()

services.function.perform_function = perform_function
responder.connect()
responder.start()

while True:
    time.sleep(0.1)  # do some other stuff
    gps.update()  # update housekeeping data
    set_param(PARAM_MODE, gps.mode)
    set_param(PARAM_DATA_VALID, gps.data_valid)
    set_param(PARAM_UTC_TIME, gps.time)
    set_param(PARAM_UTC_DATE, gps.date)
    set_param(PARAM_LAT, gps.latitude)
    set_param(PARAM_LON, gps.longitude)

responder.stop()
responder.disconnect()
```

```python
class GpsUnit:
    MODE_STANDBY = 0
    MODE_NORMAL = 1

    def __init__(self, uart):
        self.uart = uart
        self.data_valid = 0
        self.mode = self.MODE_NORMAL
        self.time = 999999
        self.date = 999999
        self.latitude = 999.0
        self.longitude = 999.0

    def send_command(self, command):
        checksum = 0
        for char in command:
            checksum ^= char
        self.uart.write(b"$" + command + b"*")
        self.uart.write(bytes(f"{checksum:02x}".upper(), "ascii"))
        self.uart.write(b"\r\n")

    def init(self):
        # configure to only get RMC messages
        self.send_command(b"PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0")

    def warm_restart(self):
        self.send_command(b"PMTK101")
        self.mode = self.MODE_NORMAL

    def cold_restart(self):
        self.send_command(b"PMTK103")
        self.mode = self.MODE_NORMAL

    def mode_standby(self):
        self.send_command(b"PMTK161,0")
        self.mode = self.MODE_STANDBY
        self.data_valid = 0

    def mode_normal(self):
        self.send_command(b"PMTK000")
        self.mode = self.MODE_NORMAL

    def update(self):
        while self.uart.any() and self.mode == self.MODE_NORMAL:
            line = self.uart.readline()
            if line:
                try:
                    line = str(line, "ascii").split(",")
                except Exception as e:
                    print(e); continue
                if line[0][3:] == "RMC":
                    self.data_valid = 1 if line[2] == "A" else 0
                    if self.data_valid:
                        self.time = int(float(line[1]))
                        self.date = int(line[9])
                        latitude = float(line[3]) / 100
                        latitude = -latitude if line[4] == "S" else latitude
                        longitude = float(line[5]) / 100
                        longitude = -longitude if line[6] == "W" else longitude
                        self.latitude = latitude
                        self.longitude = longitude
```

6. Testing (1/4)

- For the testing, we will use the laptop to run SpaceCAN Tester (which runs a controller node) and interfaces the Pyboard Prototype Module via CAN bus.
- For this we need a USB-CAN adapter. We will use the candleLight FD USB-CAN adapter (https://linux-automation.com/en/products/candlelight-fd.html) as it is fully open source and easy to use.
- The setup would be as shown below. If you only have one adapter (as for this simple demo) it will work as well, omitting the redundancy aspect.

USB

CAN to CAN_A

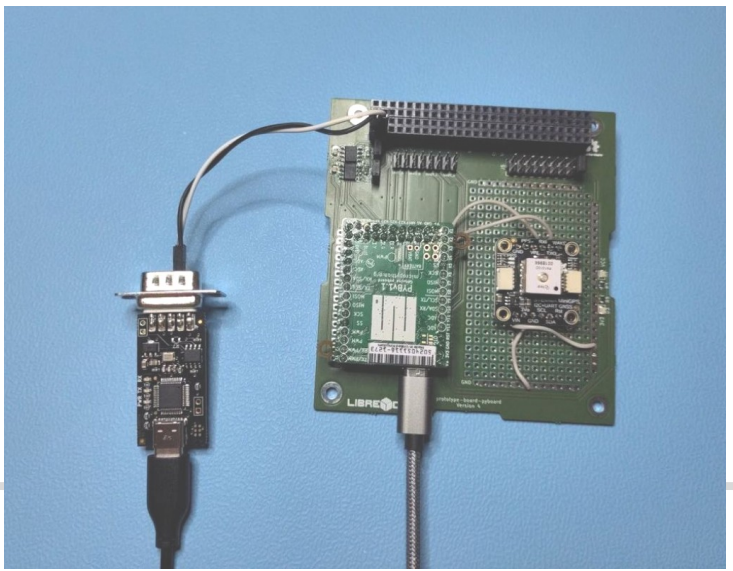
USB

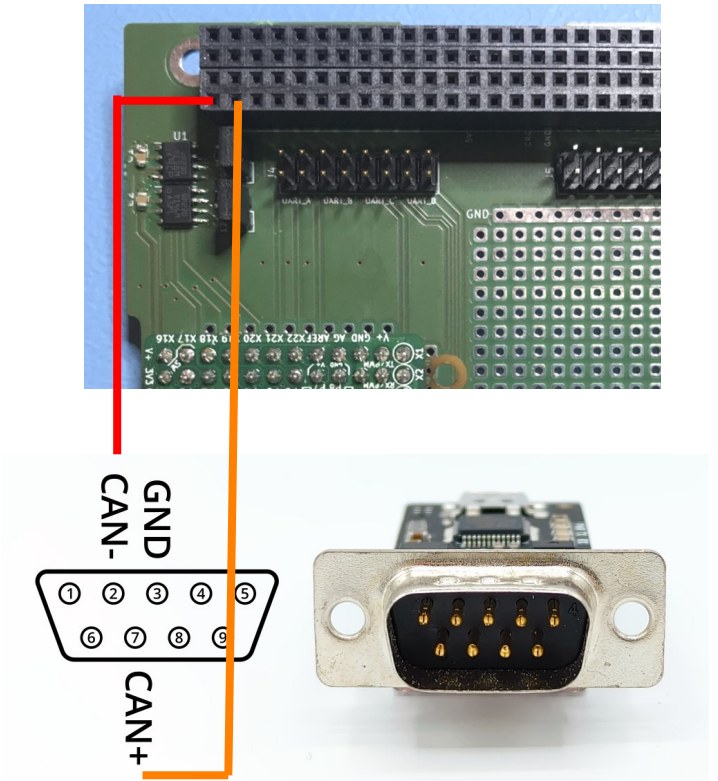
CAN to CAN_B

(optional)

6. Testing (2/4)

- You only need to connect the CAN bus related signals from adapter to the board.
- Refer to the LibreCube board specification to locate the CAN pins.
  https://librecube.gitlab.io/standards/board_specification/#electrical-specification





6. Testing (3/4)

LIBRECUBE
open source space exploration

Now we need to start the SpaceCAN Tester and the code of the pyboard:

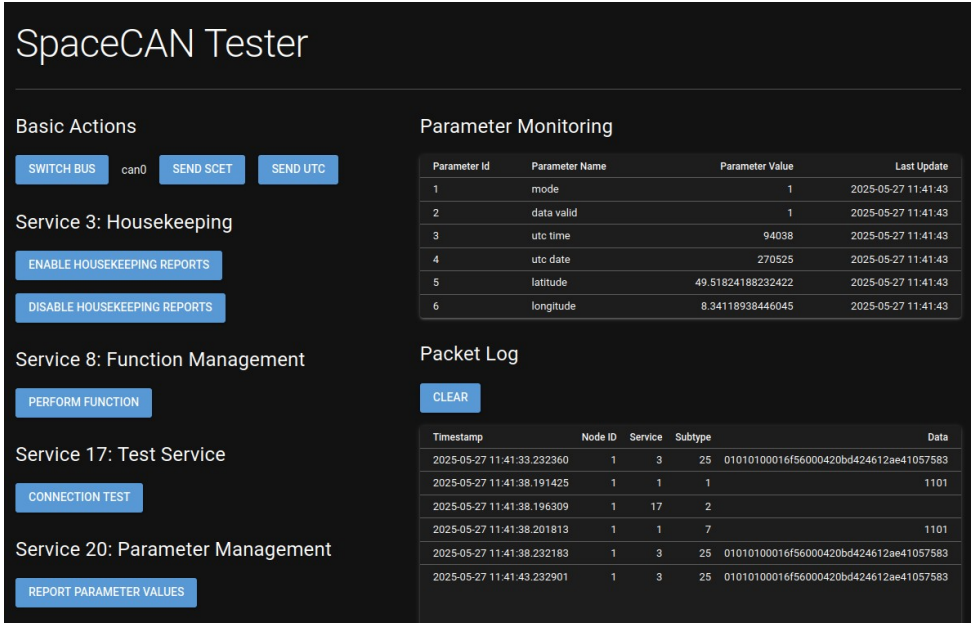
- The pyboard code will run automatically when the board is supplied with power. (Nonetheless, you can also use Thonny IDE or another serial terminal to connect to it and start it by hand.)
- For the SpaceCAN Tester, we do the following:
  - Modify the “config.json” file for the physical CAN bus.
  - Bring up the CAN bus:
    `sudo ip link set dev can0 up type can bitrate 1000000`
  - Start SpaceCAN Tester:
    `source venv/bin/activate`
    `spacecan-tester config.json`

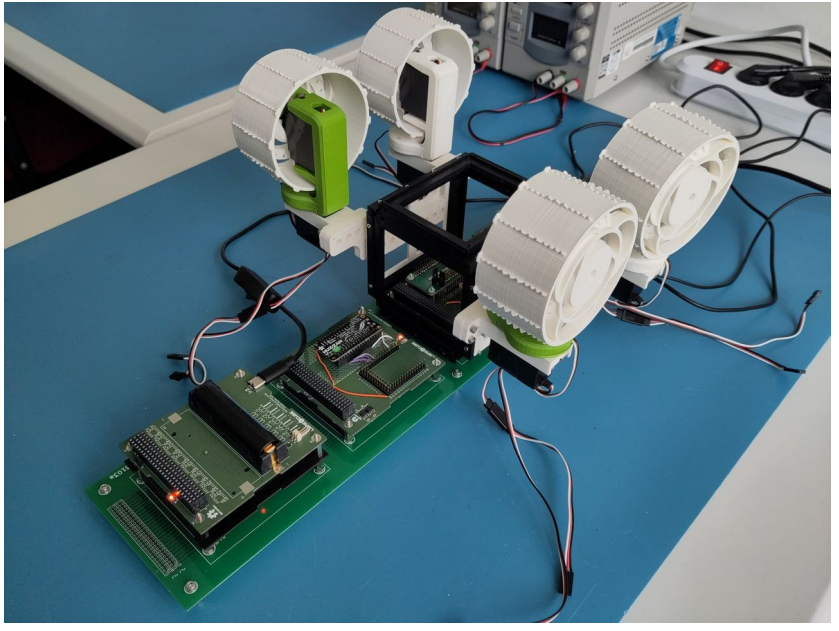
```
{
  "interface": "socketcan",
  "channel_a": "can0",
  "channel_b": "can0",
  "node_id": 1,
  "heartbeat_period": 0.5,
  "max_miss_heartbeat": 5,
  "use_packets": true,

```

6. Testing (4/4)

LIBRECUBE
open source space exploration

Now you can test all the functionality of your module using the SpaceCAN interface. As a next step you may want to integrate your module into the system you intend to develop.





Conclusion

LIBRE CUBE
open source space exploration

- This example was focusing on functional testing only.
- The example was simple, but it covered all the steps that are required for development of any complexity.
- The development of the SpaceCAN interface and the functionality-specific code can be developed in parallel and is largely independent from each other.
- Due to its modular approach, one could swap the GPS unit with a different version (eg. a Galileo receiver) while retaining the same interface to the external controller. Only the GPS related code would need to be modified.
- In general, the SpaceCAN interface code will be very similar for any kind of module, and only the code for the selected functionality-providing component will be specific. This increases maintainability and exchange.