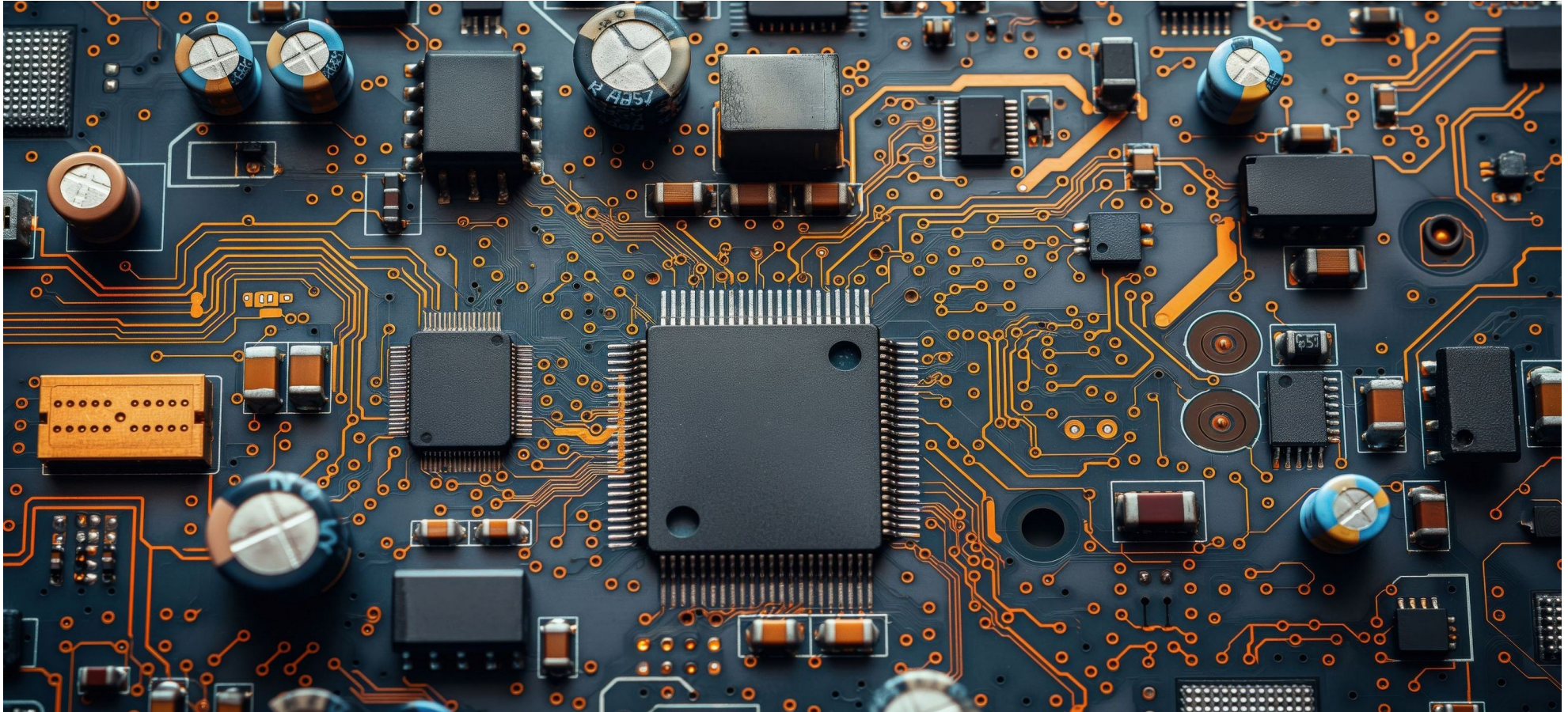


Module Development

Release date: 2025-05-27

LIBRE CUBE
open source space exploration

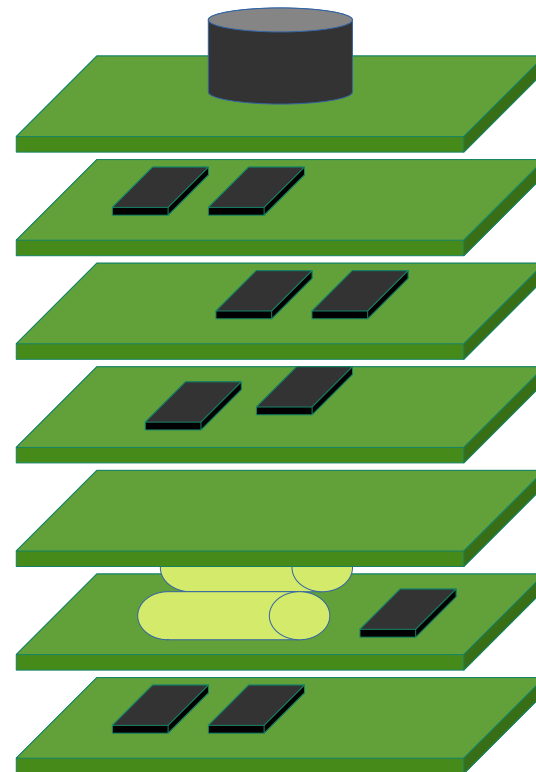
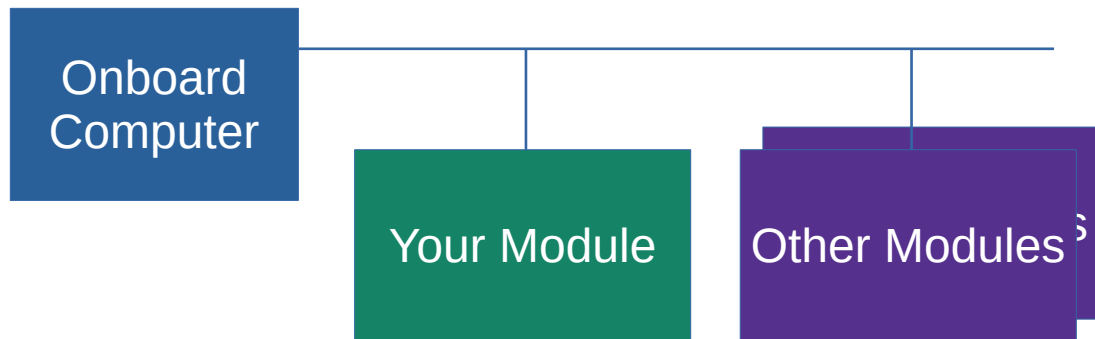


- This tutorial shows you how to develop a module that can be integrated into a system, such as a rover, satellite, drone, etc.
- You define the functionality of the module.
 - It can be a full-blown subsystem (ie. provide electrical power, propulsion, navigation) or any kind of sensor or actuator relevant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



Module Stack

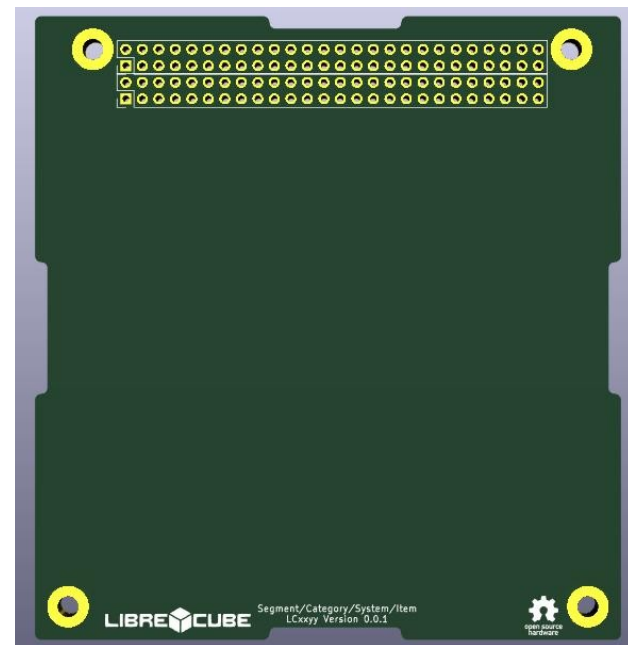
- The system (rover, etc.) will consist of a stack of modules (electronic boards) that in total provide all the functionality for the mission.
- Typically a central onboard computer monitors and controls all the modules in the stack.



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/



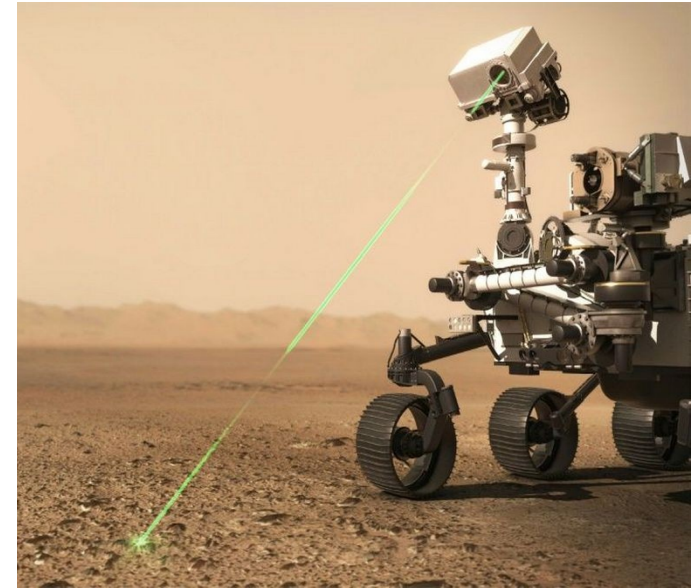
Development Workflow

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

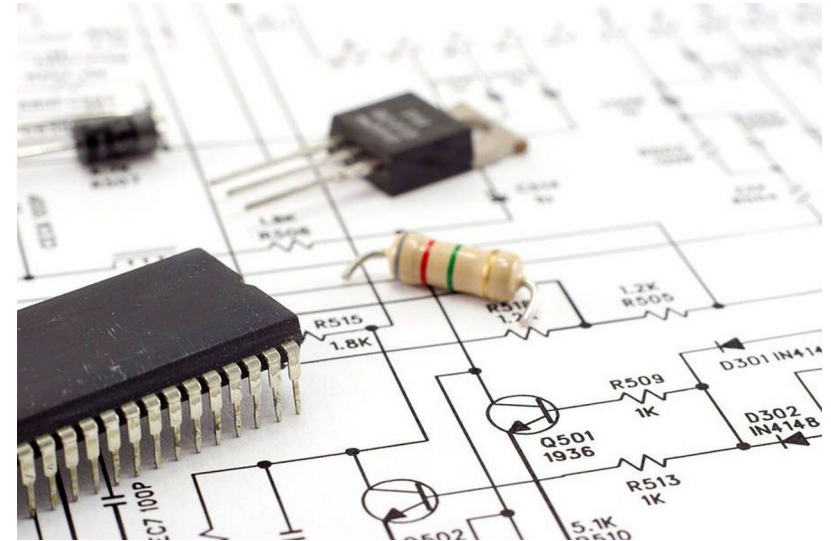
1. Define the Functionality

- 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



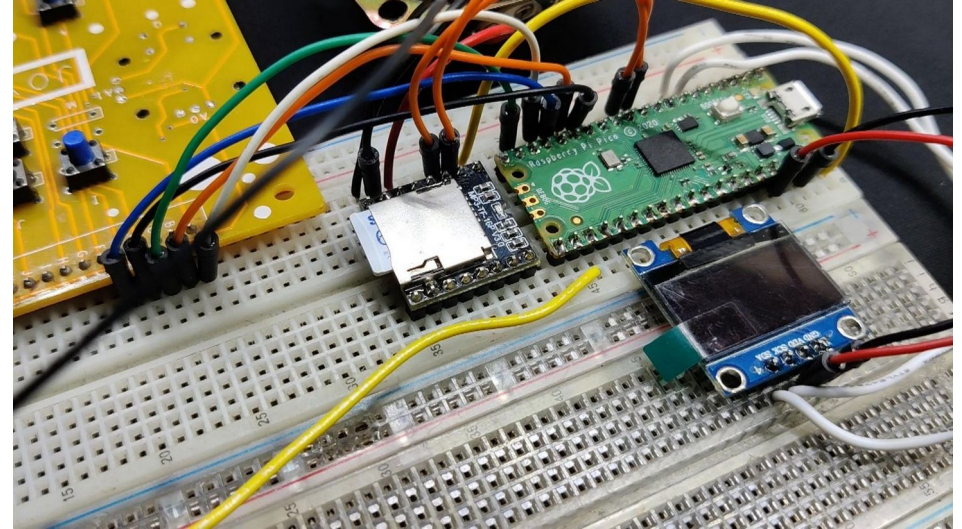
2. Design and Hardware

- 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

- 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.

SpaceCAN Tester

Basic Actions

SWITCH BUS vcan0 SEND SCOT SEND UTC

Service 3: Housekeeping

ENABLE HOUSEKEEPING REPORTS DISABLE HOUSEKEEPING REPORTS

Service 8: Function Management

PERFORM FUNCTION

Service 17: Test Service

CONNECTION TEST

Service 20: Parameter Management

REPORT PARAMETER VALUES

Parameter Monitoring

Parameter Id	Parameter Name	Parameter Value	Last Update
1	Counter	207	2025-05-16 17:22:18
2	LED	0	2025-05-16 17:22:18
3	Temperature	25.402753829956055	2025-05-16 17:22:18

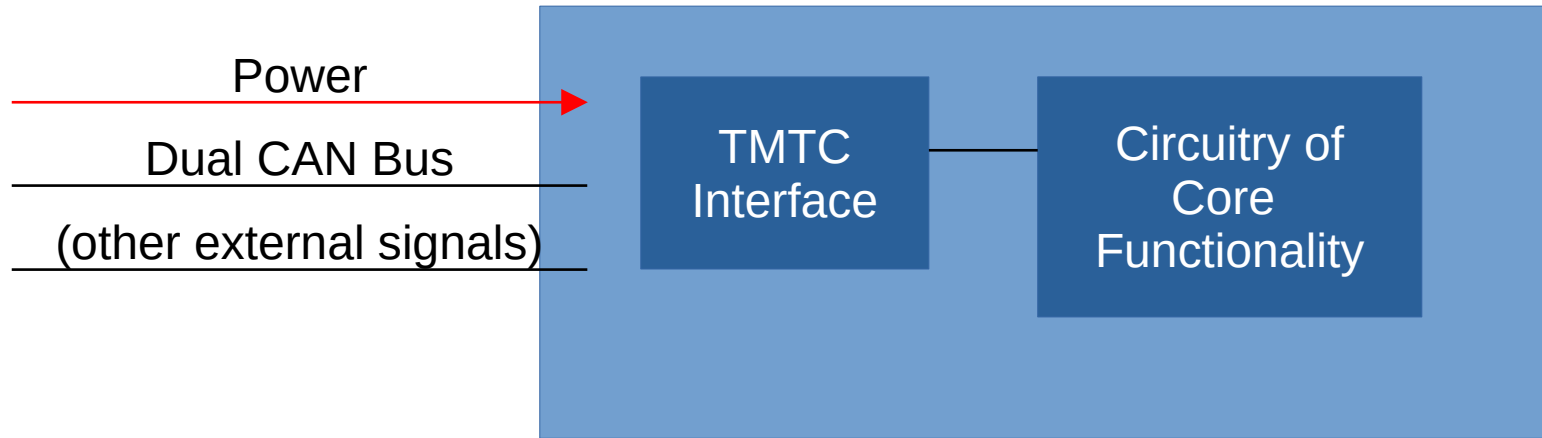
Packet Log

CLEAR

Timestamp	Node ID	Service	Subtype	Data
2025-05-16 17:21:58.061819	1	3	25	01000000080041c0e0df
2025-05-16 17:22:03.066403	1	3	25	010000003a0041c9f2bf
2025-05-16 17:22:08.069978	1	3	25	010000006c0041cf56b5
2025-05-16 17:22:13.072895	1	3	25	010000009e0041c9f34d
2025-05-16 17:22:18.075176	1	3	25	01000000cf0041cb38d7

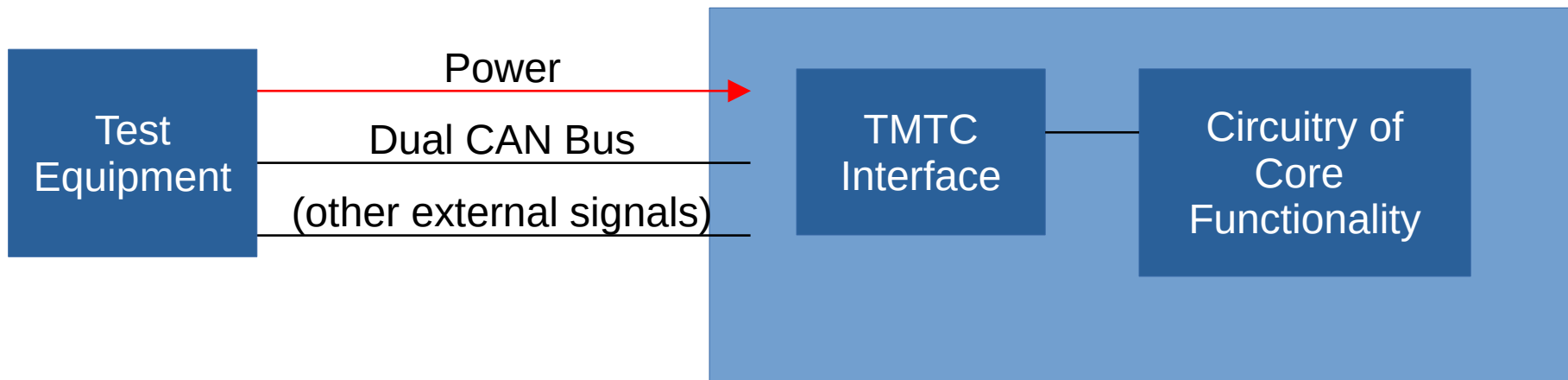
5. Integration

- 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



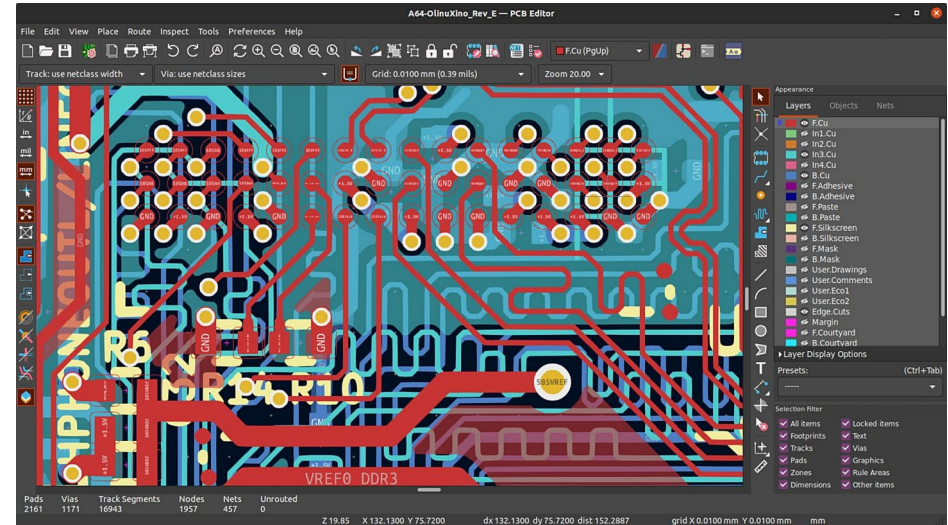
6. Testing

- 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

- 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)

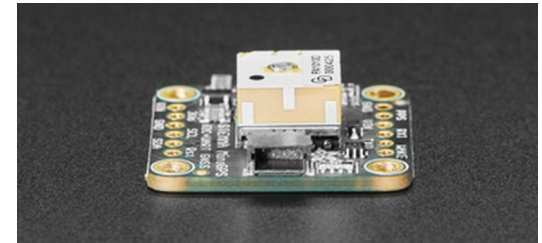
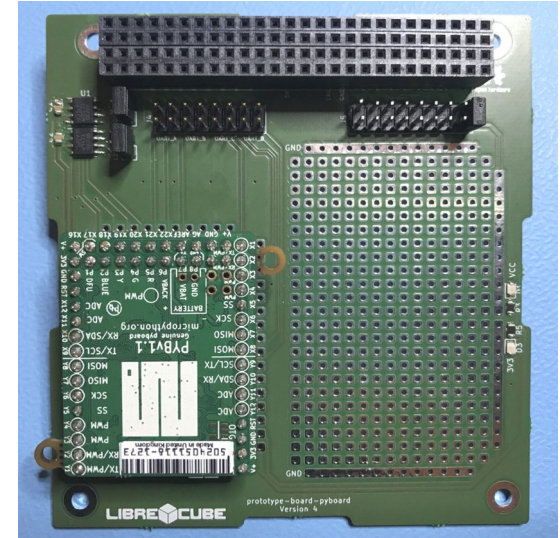
2. Design and Hardware (1/4)

- 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>

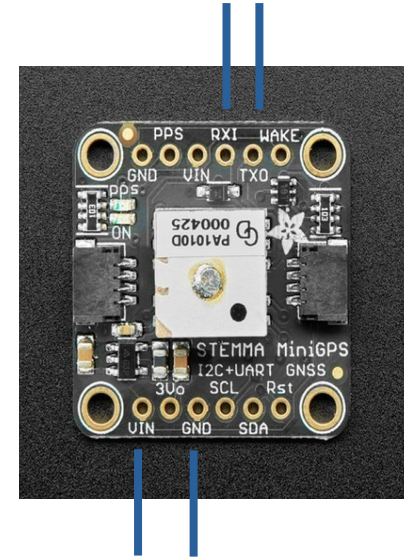
- For the GPS receiver unit we choose the Adafruit Mini GPS PA1010D.

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



2. Design and Hardware (2/4)

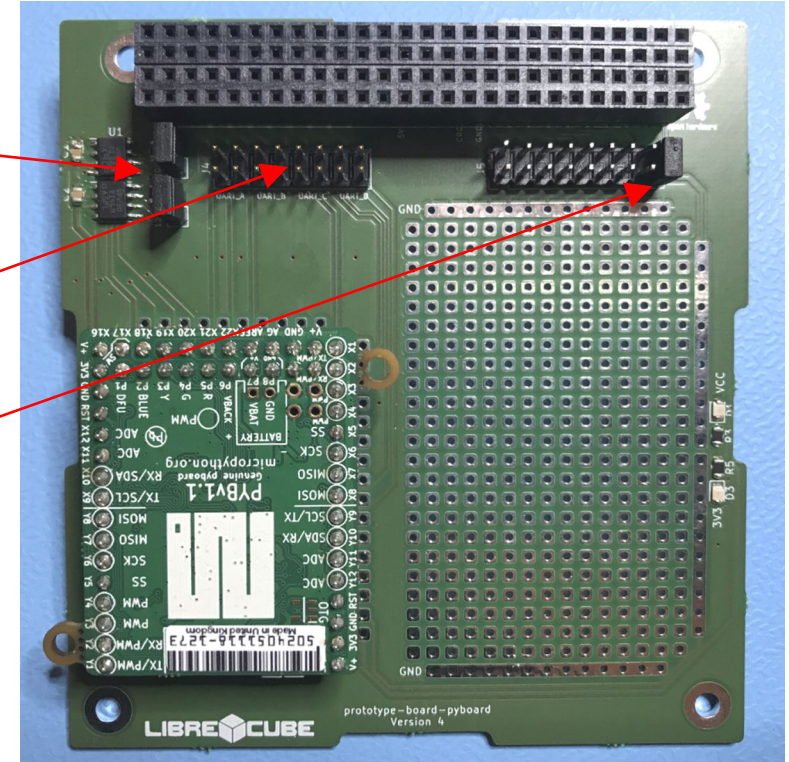
- According to the datasheet of the GPS unit [1] we will need to connect the following pins:
 - GND
 - VIN: 3.3 Volt input
 - TXO: UART TX out
 - RXI: UART RX in



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

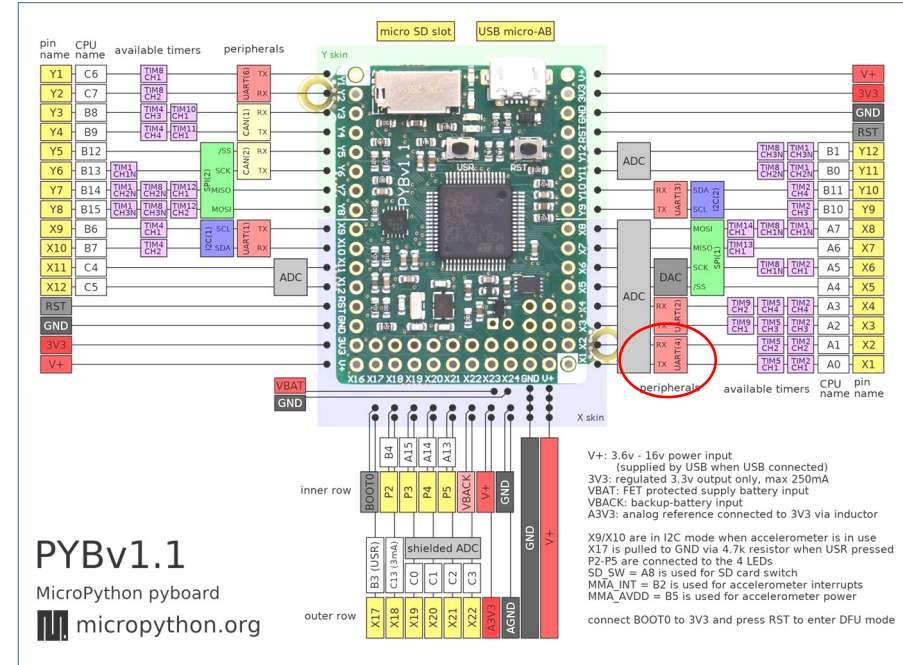
2. Design and Hardware (3/4)

- 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.



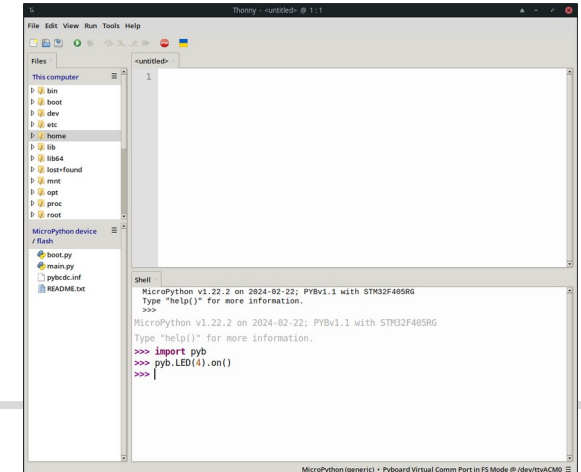
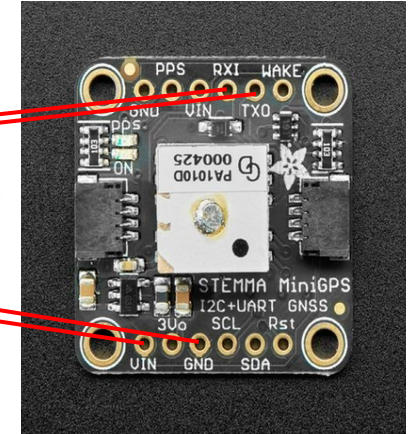
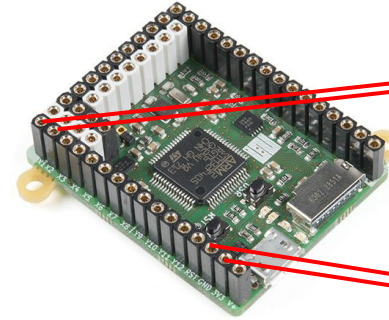
2. Design and Hardware (4/4)

- 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).



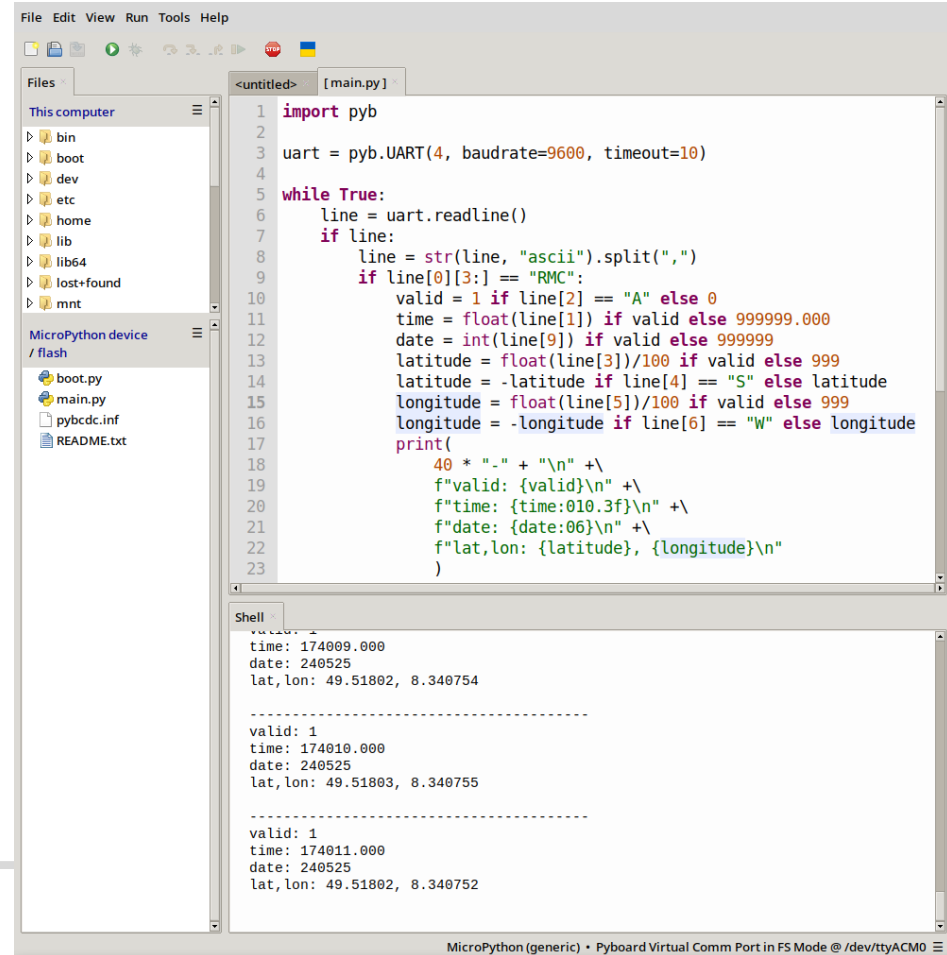
3. Breadboard Development (1/3)

- 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)

- 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.



```
File Edit View Run Tools Help
<untitled> [main.py]
1 import pyb
2
3 uart = pyb.UART(4, baudrate=9600, timeout=10)
4
5 while True:
6     line = uart.readline()
7     if line:
8         line = str(line, "ascii").split(",")
9         if line[0][3:] == "RMC":
10             valid = 1 if line[2] == "A" else 0
11             time = float(line[1]) if valid else 999999.000
12             date = int(line[9]) if valid else 999999
13             latitude = float(line[3])/100 if valid else 999
14             latitude = -latitude if line[4] == "S" else latitude
15             longitude = float(line[5])/100 if valid else 999
16             longitude = -longitude if line[6] == "W" else longitude
17             print(
18                 40 * "-" + "\n" + \
19                 f"valid: {valid}\n" + \
20                 f"time: {time:010.3f}\n" + \
21                 f"date: {date:06}\n" + \
22                 f"lat,lon: {latitude}, {longitude}\n"
23             )
24
Shell
time: 174009.000
date: 240525
lat,lon: 49.51802, 8.340754

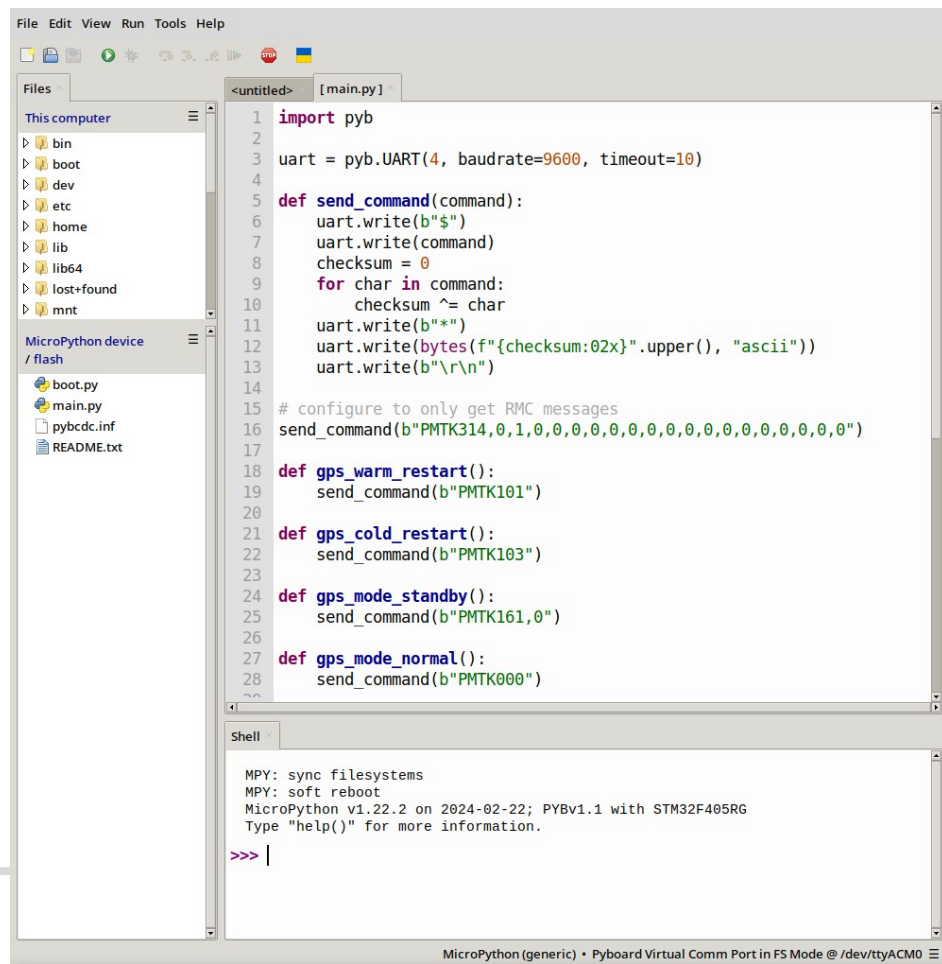
-----
valid: 1
time: 174010.000
date: 240525
lat,lon: 49.51803, 8.340755

-----
valid: 1
time: 174011.000
date: 240525
lat,lon: 49.51802, 8.340752

MicroPython (generic) • Pyboard Virtual Comm Port in FS Mode @ /dev/ttyACM0
```

3. Breadboard Development (3/3)

- 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).



```
File Edit View Run Tools Help
<untitled> [main.py]
1 import pyb
2
3 uart = pyb.UART(4, baudrate=9600, timeout=10)
4
5 def send_command(command):
6     uart.write(b"$")
7     uart.write(command)
8     checksum = 0
9     for char in command:
10         checksum ^= char
11     uart.write(b"*")
12     uart.write(bytes(f"{checksum:02x}".upper(), "ascii"))
13     uart.write(b"\r\n")
14
15 # configure to only get RMC messages
16 send_command(b"PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0")
17
18 def gps_warm_restart():
19     send_command(b"PMTK101")
20
21 def gps_cold_restart():
22     send_command(b"PMTK103")
23
24 def gps_mode_standby():
25     send_command(b"PMTK161,0")
26
27 def gps_mode_normal():
28     send_command(b"PMTK000")
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

MPY: sync filesystems
MPY: soft reboot
MicroPython V1.22.2 on 2024-02-22; PyBv1.1 with STM32F405RG
Type "help()" for more information.

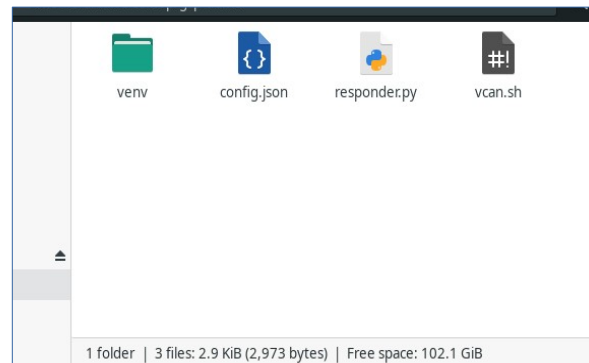
>>> |

MicroPython (generic) • Pyboard Virtual Comm Port in FS Mode @ /dev/ttyACM0

4. Prototype SpaceCAN Interface (1/4)

- 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”, “vcn.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)

- 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.

```
8  "use_packets": true,  
9  
10 "parameters": [  
11   {  
12     "parameter_id": 1,  
13     "parameter_name": "mode",  
14     "encoding": "B",  
15     "value": 1  
16   },  
17   {  
18     "parameter_id": 2,  
19     "parameter_name": "data valid",  
20     "encoding": "B",  
21     "value": 0  
22   },  
23   {  
24     "parameter_id": 3,  
25     "parameter_name": "utc time",  
26     "encoding": "L",  
27     "value": 999999  
28   },  
29   {  
30     "parameter_id": 4,  
31     "parameter_name": "utc date",  
32     "encoding": "L",  
33     "value": 999999  
34   },  
35   {  
36     "parameter_id": 5,  
37     "parameter_name": "latitude",  
38     "encoding": "f",  
39     "value": 999.0  
40   },  
41   {  
42     "parameter_id": 6,  
43     "parameter_name": "longitude",  
44     "encoding": "f",  
45     "value": 999.0  
46   }  
47 ],  
48
```

```
"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)

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

```
File Edit Search View Document Help
1 from datetime import datetime
2 import time
3 import random
4 import spacecan
5
6 responder = spacecan.Responder.from_file("config.json")
7 services = spacecan.Services(responder).from_file("config.json")
8 set_param = services.parameter.set_parameter_value
9 get_param = services.parameter.get_parameter_value
10
11 PARAM_DATA_VALID = 1
12 PARAM_MODE = 2
13 PARAM_UTC_TIME = 3
14 PARAM_UTC_DATE = 4
15 PARAM_LAT = 5
16 PARAM_LON = 6
17
18 FUNC_SET_MODE = 1
19 FUNC_SET_MODE_ARG_MODE = 1
20 FUNC_WARM_RESTART = 2
21 FUNC_COLD_RESTART = 3
22
23
24 def perform_function(function_id, arguments):
25     if function_id == FUNC_SET_MODE:
26         print("Execute: set mode")
27         if arguments[FUNC_SET_MODE_ARG_MODE] == 0:
28             set_param(PARAM_MODE, 0)
29             set_param(PARAM_DATA_VALID, 0)
30         elif arguments[FUNC_SET_MODE_ARG_MODE] == 1:
31             set_param(PARAM_MODE, 1)
32             set_param(PARAM_DATA_VALID, 1)
33     elif function_id == FUNC_WARM_RESTART:
34         print("Execute: warm restart")
35         set_param(PARAM_MODE, 1)
36     elif function_id == FUNC_COLD_RESTART:
37         print("Execute: cold restart")
38         set_param(PARAM_MODE, 1)
39
```

```
36 elif function_id == FUNC_COLD_RESTART:
37     print("Execute: cold restart")
38     set_param(PARAM_MODE, 1)
39
40
41 services.function.perform_function = perform_function
42 responder.connect()
43 responder.start()
44
45 try:
46     set_param(PARAM_DATA_VALID, 1)
47     while True:
48         # do some other stuff
49         time.sleep(0.1)
50
51     # update housekeeping data
52     set_param(PARAM_UTC_TIME, int(datetime.now().strftime("%H%M%S")))
53     set_param(PARAM_UTC_DATE, int(datetime.now().strftime("%d%m%y")))
54     set_param(PARAM_LAT, random.uniform(49, 50))
55     set_param(PARAM_LON, random.uniform(8, 9))
56
57 except KeyboardInterrupt:
58     pass
59
60 responder.stop()
61 responder.disconnect()
62 print("Stopped")
63
```

Filetype: Py

4. Prototype SpaceCAN Interface (4/4)

- 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`

SpaceCAN Tester

Basic Actions

`SWITCH BUS` vcan0 `SEND SCET` `SEND UTC`

Service 3: Housekeeping

`ENABLE HOUSEKEEPING REPORTS`
`DISABLE HOUSEKEEPING REPORTS`

Service 8: Function Management

`PERFORM FUNCTION`

Service 17: Test Service

`CONNECTION TEST`

Service 20: Parameter Management

`REPORT PARAMETER VALUES`

Parameter Monitoring

Parameter Id	Parameter Name	Parameter Value	Last Update
1	data valid	1	2025-05-24 22:45:33
2	mode	1	2025-05-24 22:45:33
3	utc time	224533	2025-05-24 22:45:33
4	utc date	240525	2025-05-24 22:45:33
5	latitude	49.51435852050781	2025-05-24 22:45:33
6	longitude	8.33290100097656	2025-05-24 22:45:33

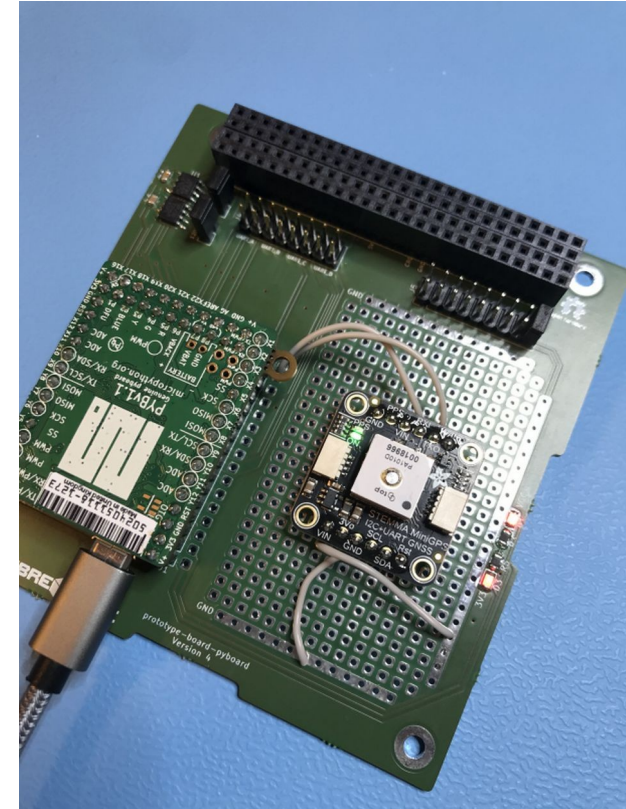
Packet Log

`CLEAR`

Timestamp	Node ID	Service	Subtype	Data
2025-05-24 22:35:08.441799	1	3	25	010101000369140003ab8d424577424107d439
2025-05-24 22:35:13.445343	1	3	25	010101000369190003ab8d4244fab410a8a72
2025-05-24 22:35:18.448607	1	3	25	0101010003691e0003ab8d4246e67a41077b15
2025-05-24 22:35:23.451788	1	3	25	010101000369230003ab8d4247895141069318
2025-05-24 22:35:28.454649	1	3	25	010101000369280003ab8d4244eb7a410d7cbb
2025-05-24 22:35:33.457275	1	3	25	0101010003692d0003ab8d42459e3d410ceec
2025-05-24 22:35:38.459748	1	3	25	010101000369320003ab8d42442711410a514f
2025-05-24 22:35:43.463091	1	3	25	010101000369370003ab8d424705e7410ce3e6

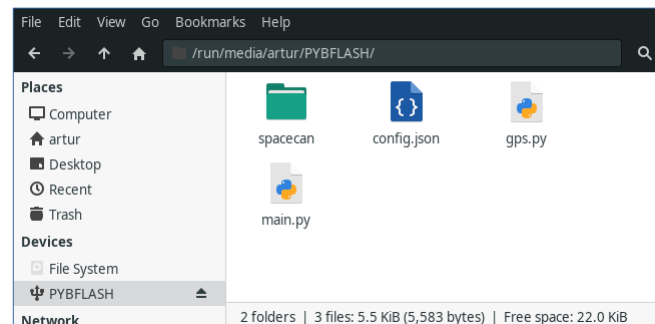
5. Integration (1/3)

- 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)

- 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.



```
1 {  
2   "interface": "pyboard",  
3   "channel_a": 1,  
4   "channel_b": 2,  
5   "node_id": 1,  
6   "heartbeat_period": 0.5,  
7   "max_miss_heartbeat": 5,  
8   "use_packets": true,  
9  
10  "parameters": [  
11    {  
12      "parameter_id": 1,  
13      "parameter_name": "data valid",  
14      "encoding": "n"
```

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.

```
1 import time
2 import pyb
3 import spacecan
4 from gps import GpsUnit
5
6 uart = pyb.UART(4, baudrate=9600, timeout=10)
7 gps = GpsUnit(uart)
8 gps.init()
9
10 responder = spacecan.Responder.from_file("config.json")
11 services = spacecan.Services(responder).from_file("config.json")
12 set_param = services.parameter.set_parameter_value
13 get_param = services.parameter.get_parameter_value
14
15 PARAM_MODE = 1
16 PARAM_DATA_VALID = 2
17 PARAM_UTC_TIME = 3
18 PARAM_UTC_DATE = 4
19 PARAM_LAT = 5
20 PARAM_LON = 6
21
22 FUNC_SET_MODE = 1
23 FUNC_SET_MODE_ARG_MODE = 1
24 FUNC_WARM_RESTART = 2
25 FUNC_COLD_RESTART = 3
26
27 def perform_function(function_id, arguments):
28     if function_id == FUNC_SET_MODE:
29         if arguments[FUNC_SET_MODE_ARG_MODE] == 0:
30             gps.mode_standby()
31         elif arguments[FUNC_SET_MODE_ARG_MODE] == 1:
32             gps.mode_normal()
33     elif function_id == FUNC_WARM_RESTART:
34         gps.warm_restart()
35     elif function_id == FUNC_COLD_RESTART:
36         gps.cold_restart()
37
38 services.function.perform_function = perform_function
39 responder.connect()
40 responder.start()
41
42 while True:
43     time.sleep(0.1) # do some other stuff
44     gps.update() # update housekeeping data
45     set_param(PARAM_MODE, gps.mode)
46     set_param(PARAM_DATA_VALID, gps.data_valid)
47     set_param(PARAM_UTC_TIME, gps.time)
48     set_param(PARAM_UTC_DATE, gps.date)
49     set_param(PARAM_LAT, gps.latitude)
50     set_param(PARAM_LON, gps.longitude)
51
52 responder.stop()
53 responder.disconnect()
54
```

```
1 class GpsUnit:
2     MODE_STANDBY = 0
3     MODE_NORMAL = 1
4
5     def __init__(self, uart):
6         self.uart = uart
7         self.data_valid = 0
8         self.mode = self.MODE_NORMAL
9         self.time = 999999
10        self.date = 999999
11        self.latitude = 999.0
12        self.longitude = 999.0
13
14    def send_command(self, command):
15        checksum = 0
16        for char in command:
17            checksum ^= char
18        self.uart.write(b"$" + command + b"*\n")
19        self.uart.write(bytes(f"{checksum:02x}").upper(), "ascii")
20        self.uart.write(b"\r\n")
21
22    def init(self):
23        # configure to only get RMC messages
24        self.send_command(b"PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0")
25
26    def warm_restart(self):
27        self.send_command(b"PMTK101")
28        self.mode = self.MODE_NORMAL
29
30    def cold_restart(self):
31        self.send_command(b"PMTK103")
32        self.mode = self.MODE_NORMAL
33
34    def mode_standby(self):
35        self.send_command(b"PMTK161,0")
36        self.mode = self.MODE_STANDBY
37        self.data_valid = 0
38
39    def mode_normal(self):
40        self.send_command(b"PMTK000")
41        self.mode = self.MODE_NORMAL
42
43    def update(self):
44        while self.uart.any() and self.mode == self.MODE_NORMAL:
45            line = self.uart.readline()
46            if line:
47                try:
48                    line = str(line, "ascii").split(",")
49                except Exception as e:
50                    print(e); continue
51                if line[0][3:] == "RMC":
52                    self.data_valid = 1 if line[2] == "A" else 0
53                    if self.data_valid:
54                        self.time = int(float(line[1]))
55                        self.date = int(line[9])
56                        latitude = float(line[3]) / 100
57                        latitude = -latitude if line[4] == "S" else latitude
58                        longitude = float(line[5]) / 100
59                        longitude = -longitude if line[6] == "W" else longitude
60                        self.latitude = latitude
61                        self.longitude = longitude
62
```

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



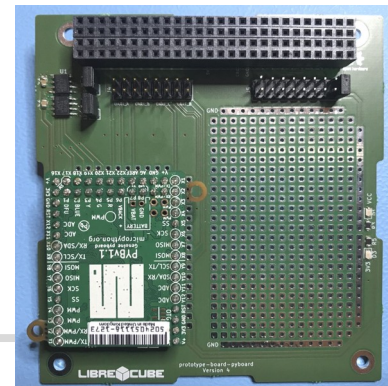
CAN to CAN_A

USB



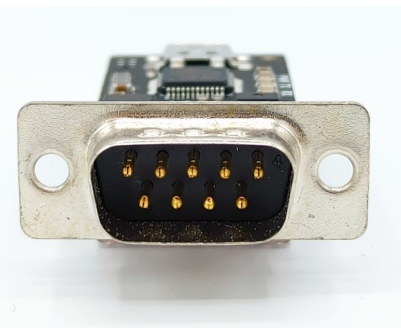
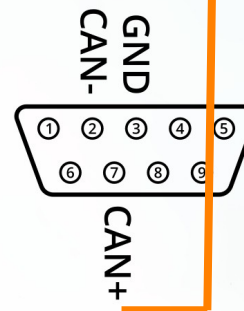
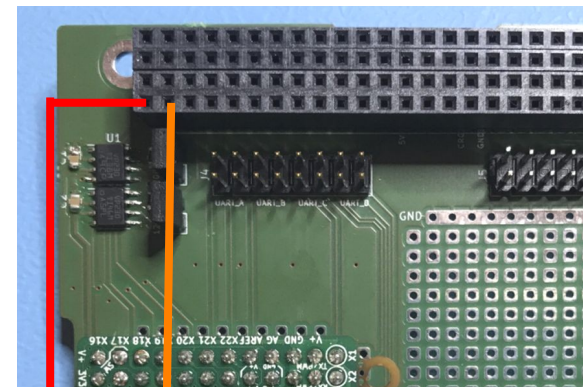
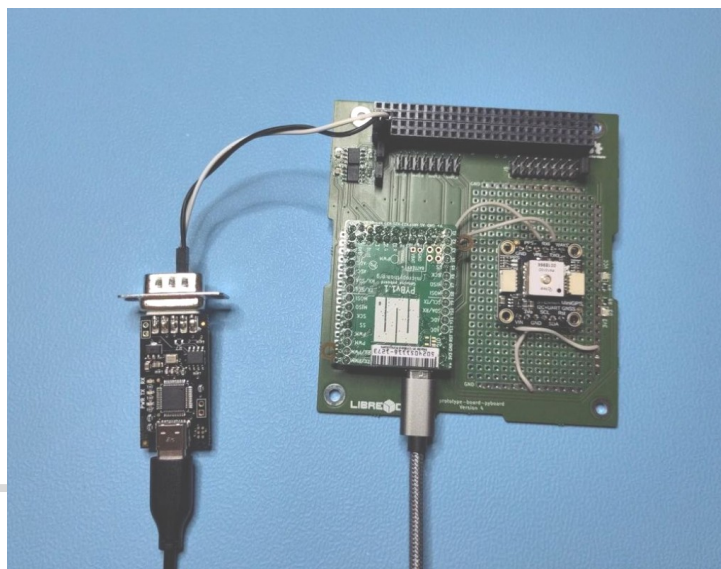
CAN to CAN_B

(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)

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`

```
1 {  
2   "interface": "socketcan",  
3   "channel_a": "can0",  
4   "channel_b": "can0",  
5   "node_id": 1,  
6   "heartbeat_period": 0.5,  
7   "max_miss_heartbeat": 5,  
8   "use_packets": true,  
9 }
```

6. Testing (4/4)

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.

SpaceCAN Tester

Basic Actions

can0

SEND SCET

SEND UTC

Service 3: Housekeeping

ENABLE HOUSEKEEPING REPORTS

DISABLE HOUSEKEEPING REPORTS

Service 8: Function Management

PERFORM FUNCTION

Service 17: Test Service

CONNECTION TEST

Service 20: Parameter Management

REPORT PARAMETER VALUES

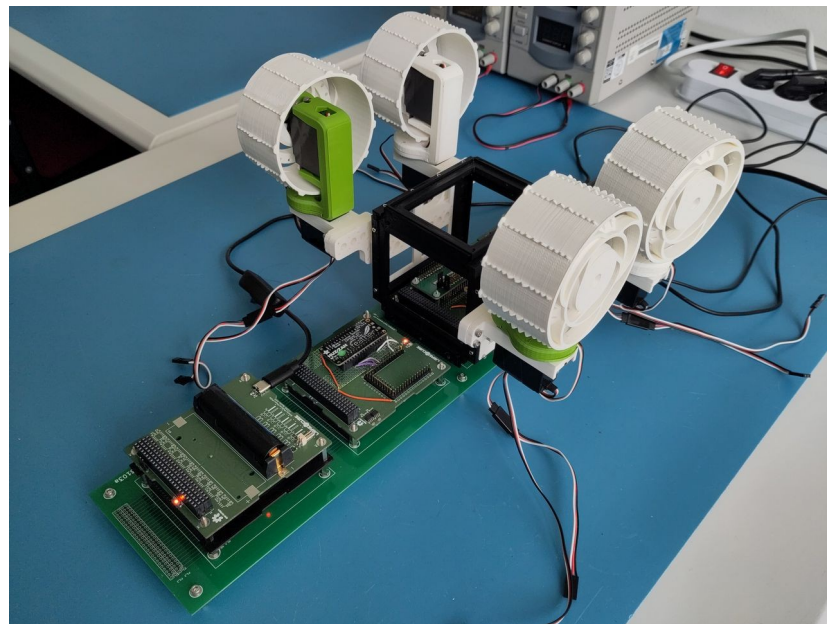
Parameter Monitoring

Parameter Id	Parameter Name	Parameter Value	Last Update
1	mode	1	2025-05-27 11:41:43
2	data valid	1	2025-05-27 11:41:43
3	utc time	94038	2025-05-27 11:41:43
4	utc date	270525	2025-05-27 11:41:43
5	latitude	49.51824188232422	2025-05-27 11:41:43
6	longitude	8.34118938446045	2025-05-27 11:41:43

Packet Log

CLEAR

Timestamp	Node ID	Service	Subtype	Data
2025-05-27 11:41:33.232360	1	3	25	01010100016f56000420bd424612ae41057583
2025-05-27 11:41:38.191425	1	1	1	1101
2025-05-27 11:41:38.196309	1	17	2	
2025-05-27 11:41:38.201813	1	1	7	1101
2025-05-27 11:41:38.232183	1	3	25	01010100016f56000420bd424612ae41057583
2025-05-27 11:41:43.232901	1	3	25	01010100016f56000420bd424612ae41057583



- 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.