

2021.05.07

# On-board software laboratory

Juan Antonio de la Puente [juan.de.la.puente@upm.es](mailto:juan.de.la.puente@upm.es)

Juan Zamorano [jzamora@datsi.fi.upm.es](mailto:jzamora@datsi.fi.upm.es)



Some rights reserved. This work is licensed under a  
[Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License..](https://creativecommons.org/licenses/by-nc-sa/4.0/)

STRAST

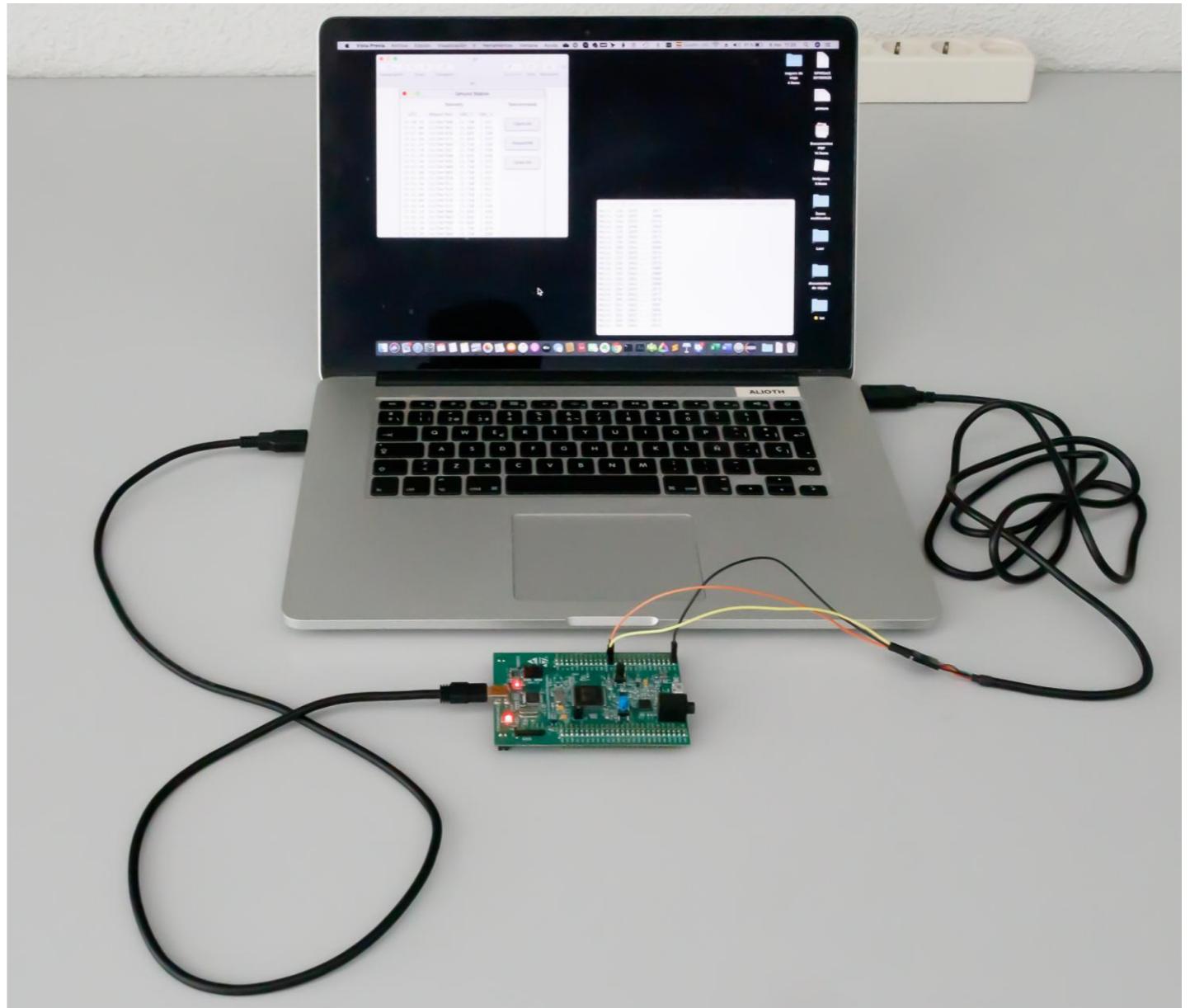
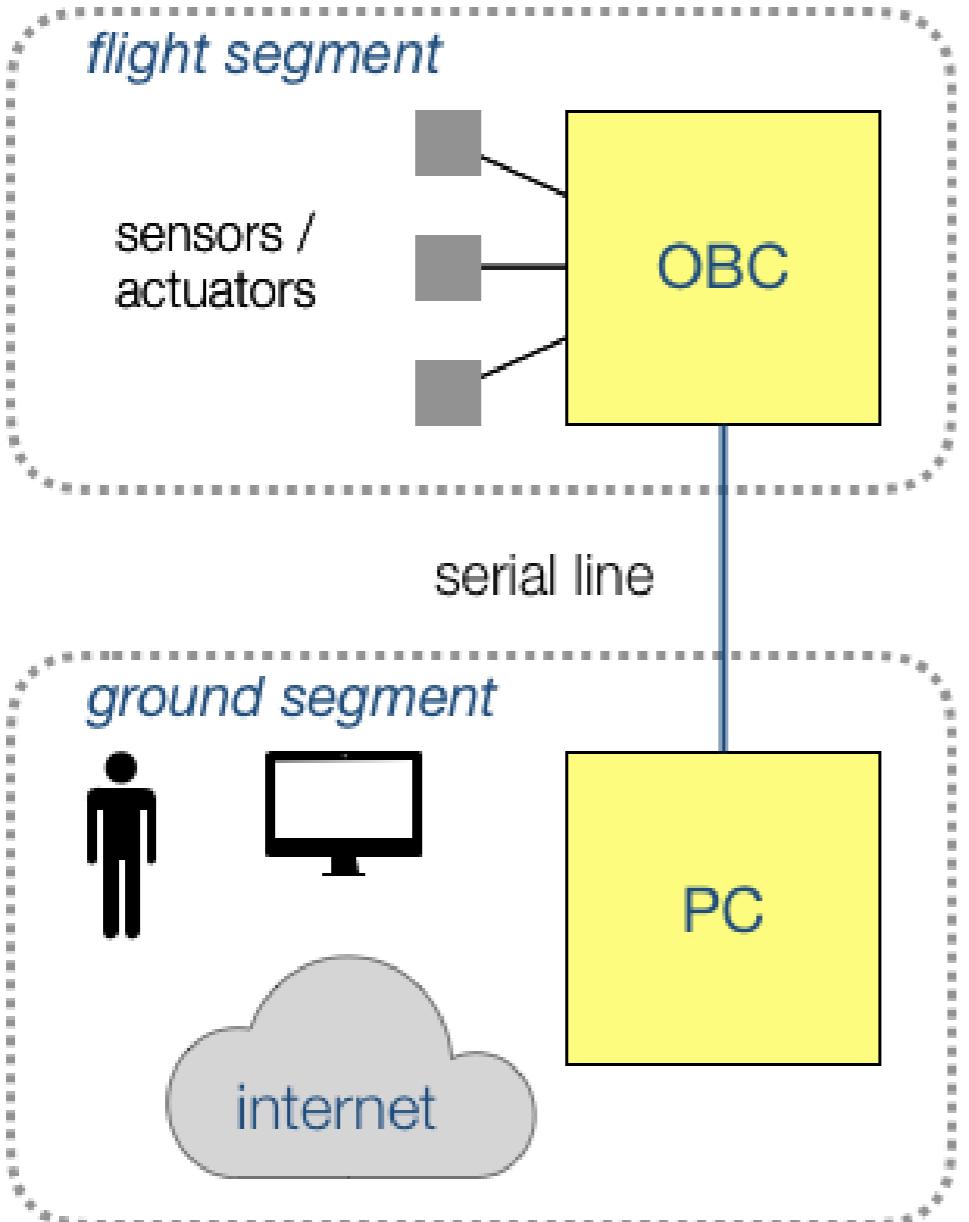
# Laboratory assignments

---

1. Installation of a native programming environment.
2. Installation of the cross-platform programming tools.
3. Simple housekeeping program.
4. Tasking program.
5. Real-time program, including temporal analysis
6. On-board data handling (OBDH) system.

# Laboratory kit

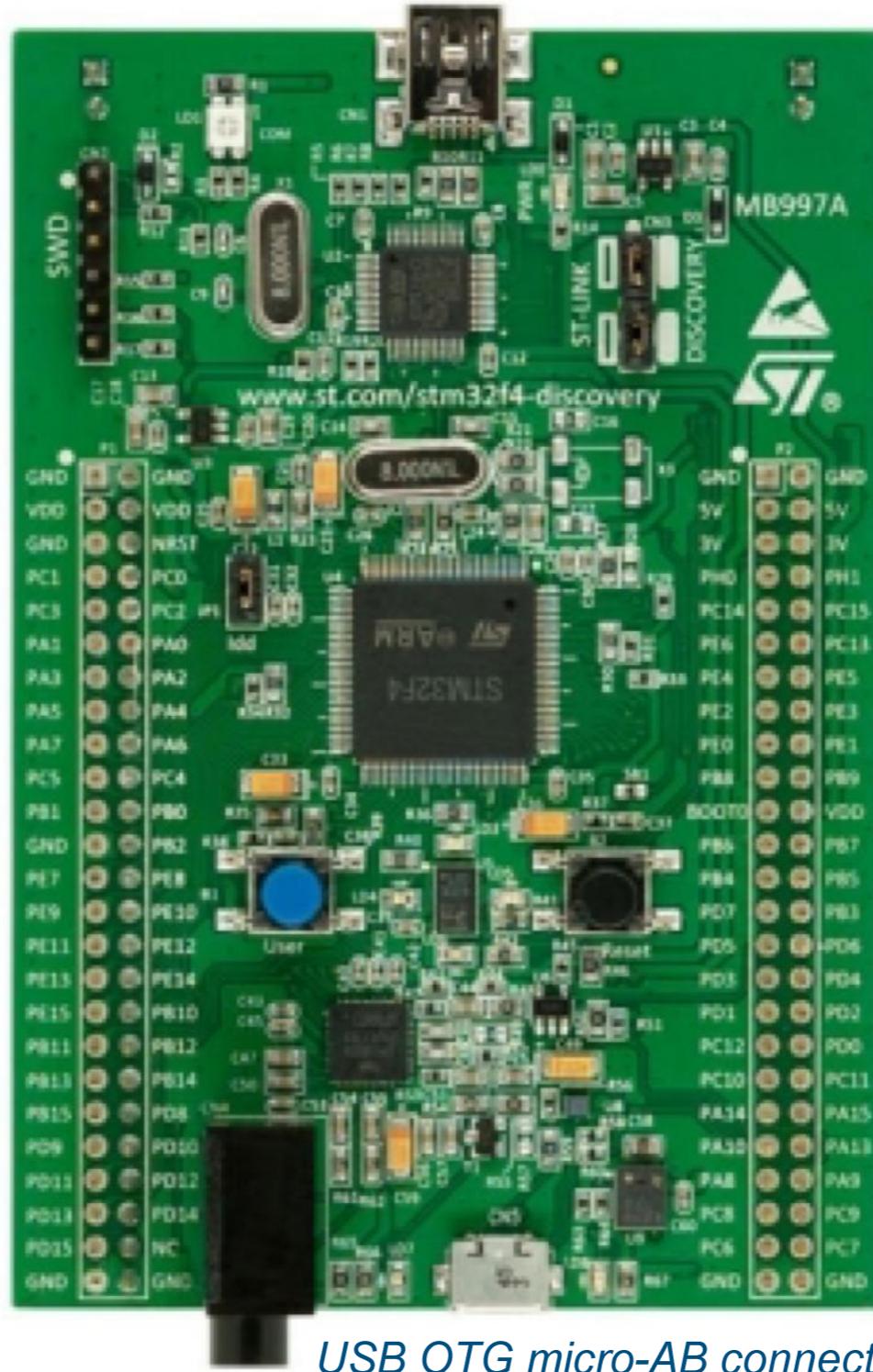
---



# Computer board

---

*USB ST-LINK mini-B connector*



*PB6 & PB7 pins*

*GND pin*

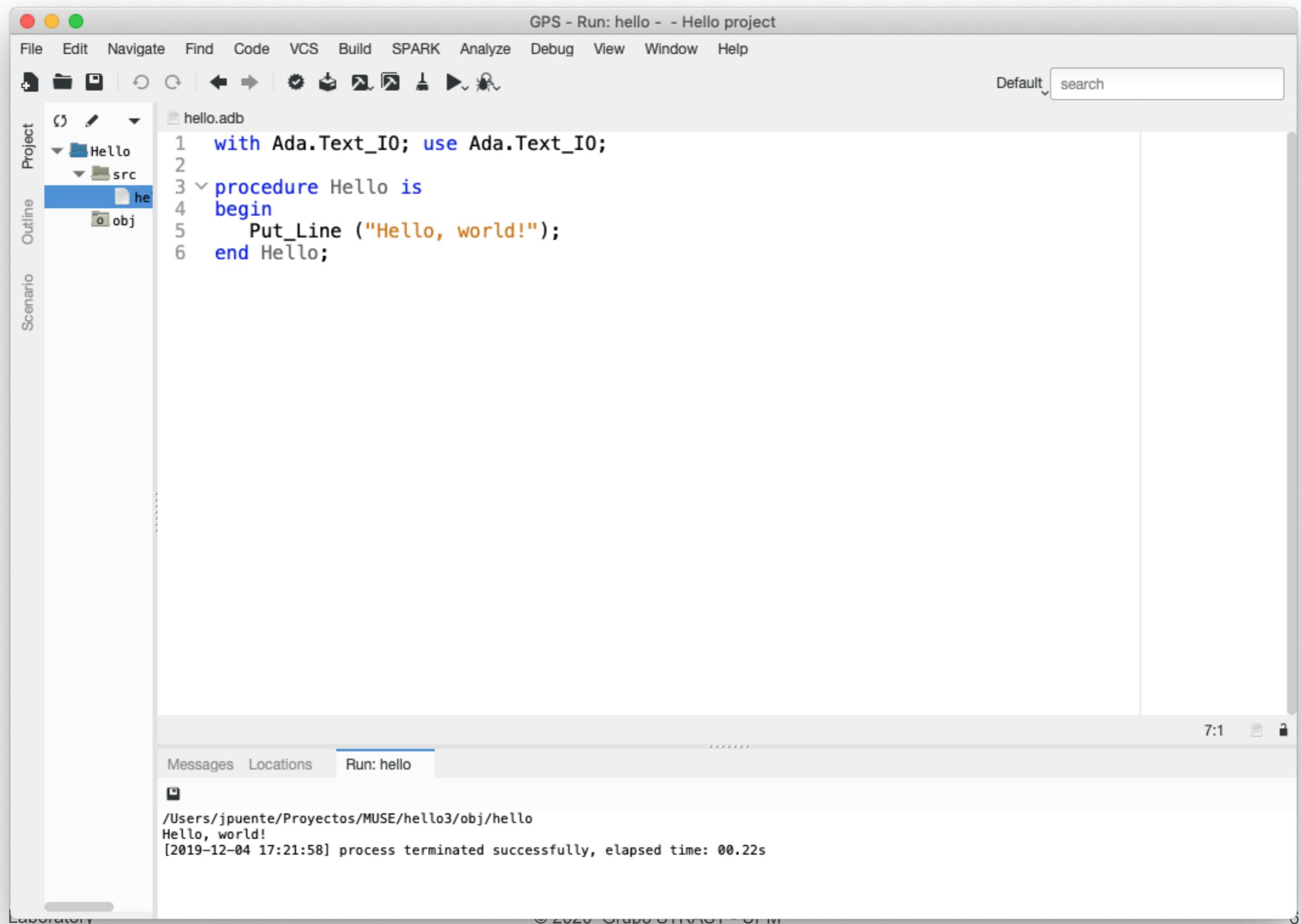
*USB OTG micro-AB connector*

# 1. Install the native compiler

---

- Windows:
  - Download and run  
gnat-2020-20200429-x86\_64-windows-bin.exe
  - Run the GPS application
  - Create a new project and enter the source code for hello.adb
  - Build and run the executable
- MacOS and Linux
  - See laboratory guide

# Sample screenshot

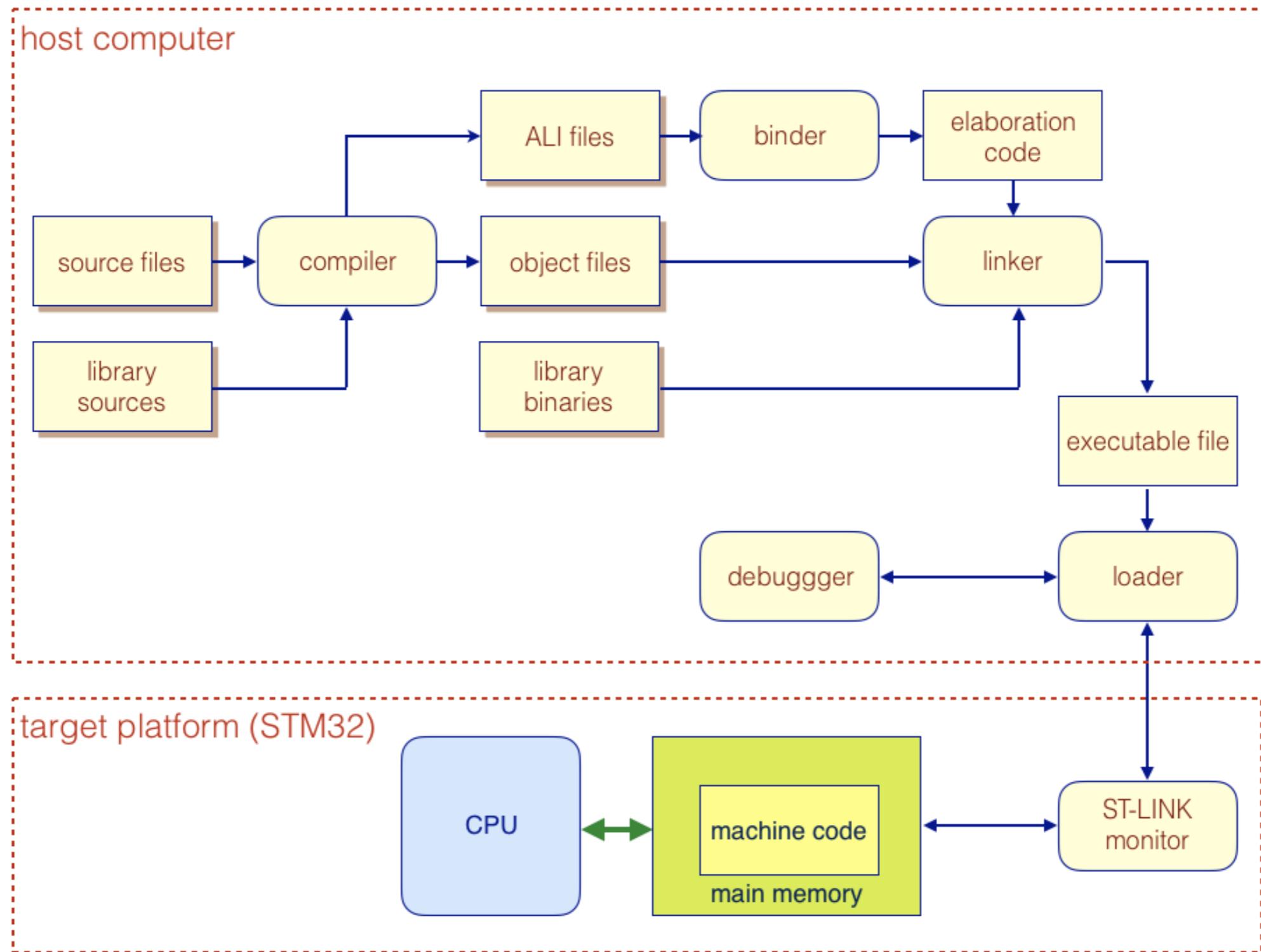


## 2a. Install cross compiler for STM32F4

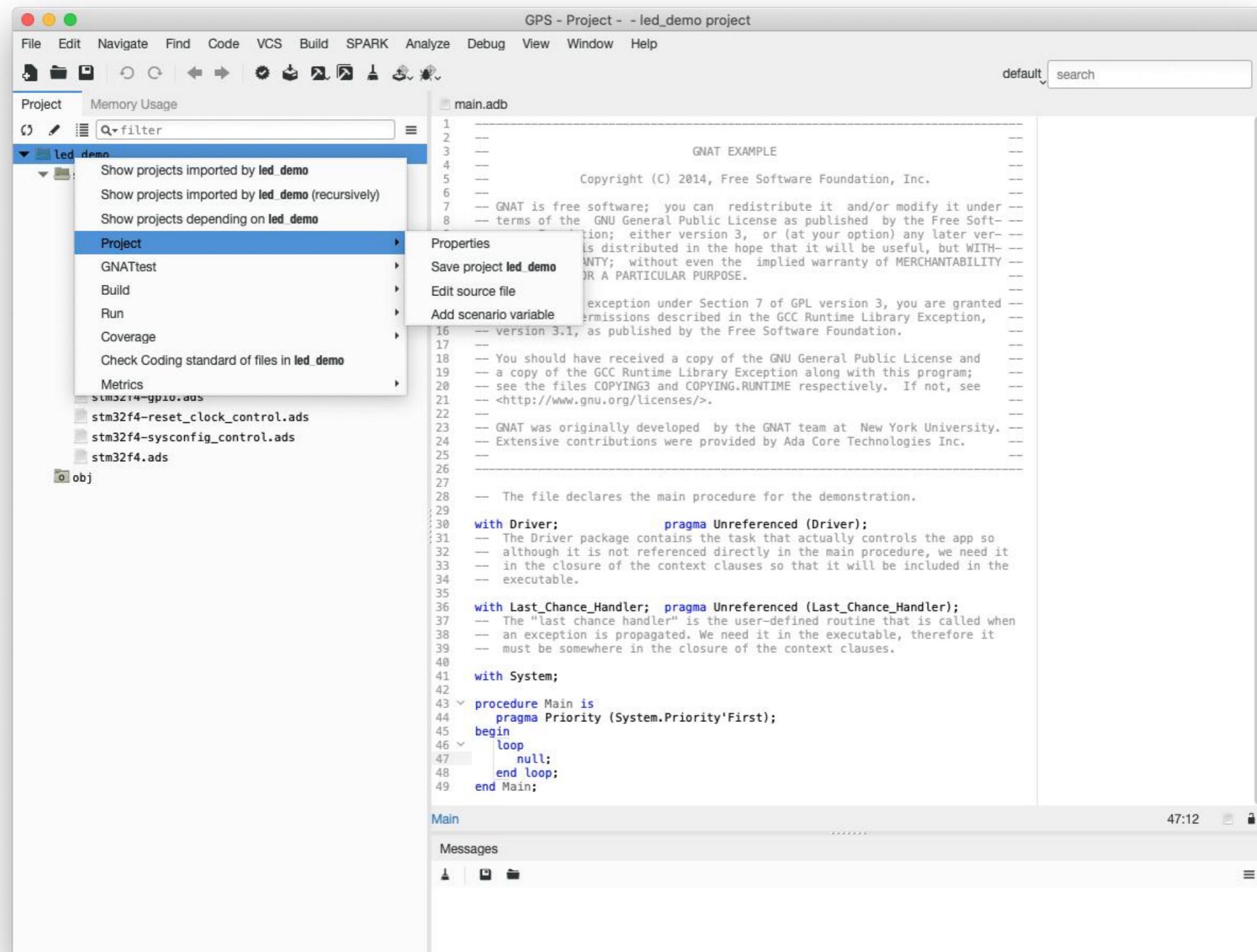
---

- Windows:
  - Download and execute the cross-compiler  
gnat-2020-20200429-arm-elf-windows64-bin.exe
  - Download and execute the ST-LINK driver
    - follow the instructions in readme.txt
  - Start the GPS application
  - Create a new project
    - File > New Project > STM324F compatible > LED demo project template
    - Set Project > Properties > Embedded > Connection tool to st-util
  - Build and run the executable
  - Connect the board to the host computer and flash the executable to the board

# Cross-compilation tools



# Sample screenshot



## 2b. Install the Ada Drivers Library

---

- All platforms:
  - Download zip file from  
[https://github.com/AdaCore/Ada\\_Drivers\\_Library](https://github.com/AdaCore/Ada_Drivers_Library)
  - Unzip the archive and move the resulting folder to your laboratory folder
  - Rename the folder to Ada\_Drivers\_Library
  - Open project  
.../Ada\_Drivers\_Library/examples/STMF4\_DISCO/blinky\_f4disco.gpr.
  - Build the executable and flash it to the board

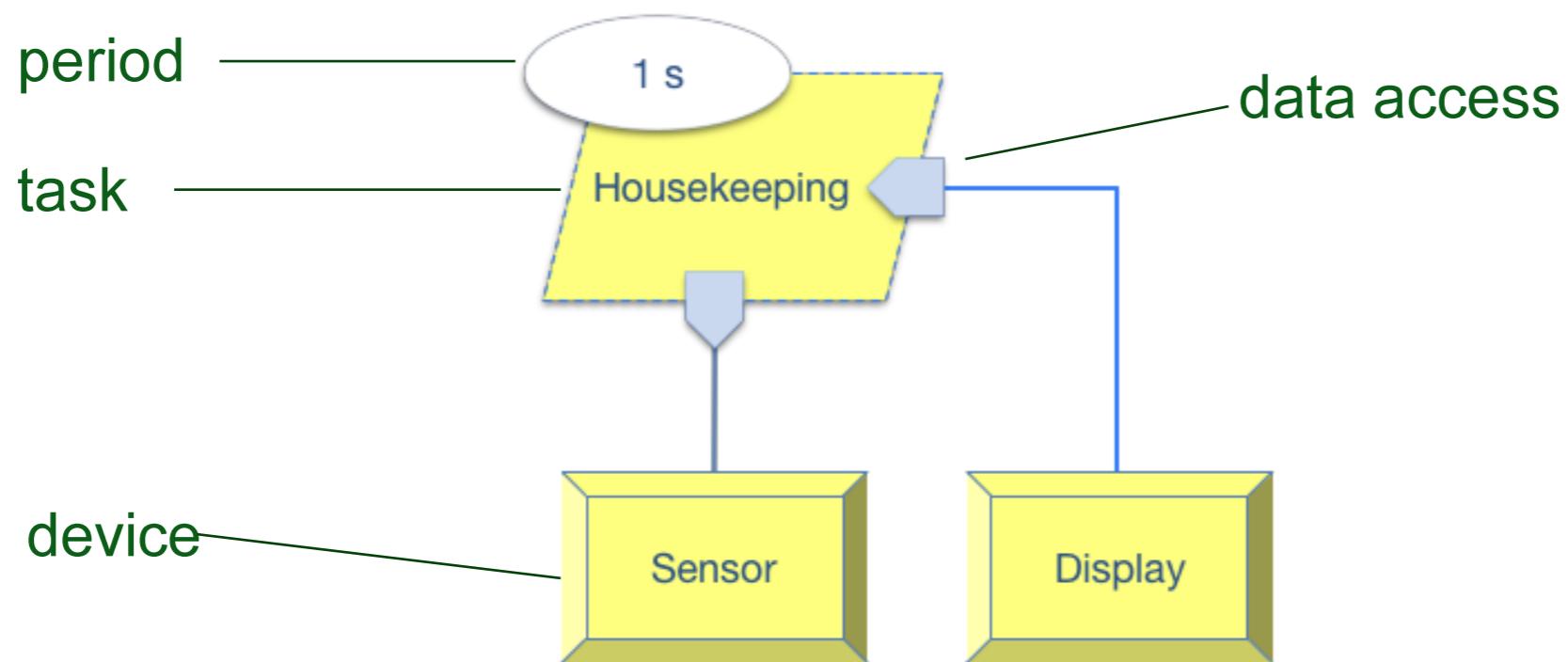
### 3. Simple housekeeping system

---

- Read a single sensor (OBC\_T) and show the reading on the screen
  - the OBC board has no screen
    - use the debugger (for now) - *semi-hosting*
  - use the Ada Drivers Library (ADL) to read the sensor
    - STM32.Device
    - STM32.ADC

# Software architecture

## AADL : Architecture Analysis & Design Language



## AADL components implemented as Ada packages

- main procedure: OBDH
- additional packages for data definitions & others as necessary

# Temperature sensor

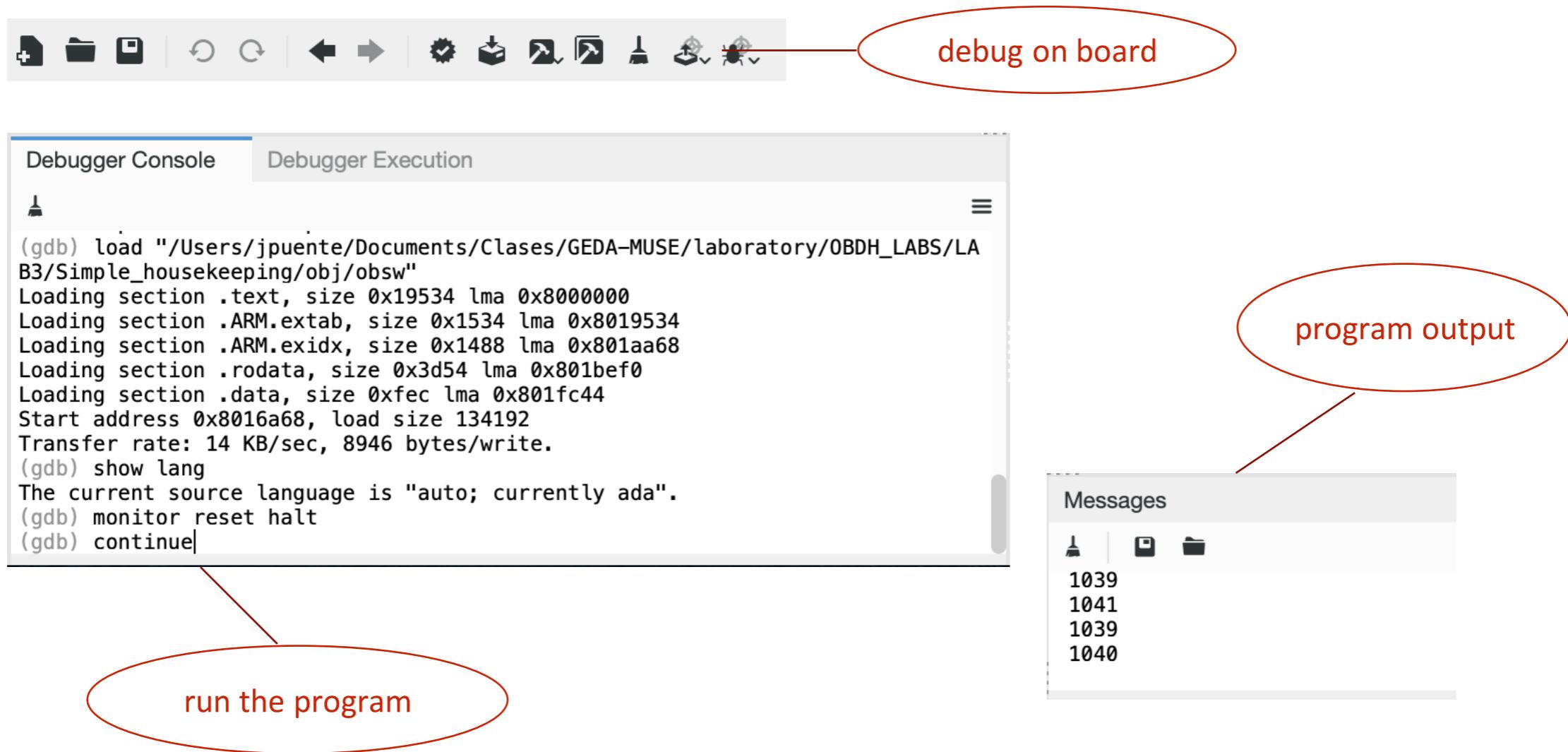
---

- **OBC\_T : MCU internal temperature sensor**
  - connected to ADC1\_IN16 input
  - range -40 .. +125 °C
  - 14-bit reading VSENSE: 0 .. 4095
  - Temperature (in °C) =  $\{(VSENSE - V25) / \text{Avg\_Slope}\} + 25$

Symbol	Parameter	Min	Typ	Max	Unit
$T_L^{(1)}$	$V_{SENSE}$ linearity with temperature	-	$\pm 1$	$\pm 2$	°C
Avg_Slope <sup>(1)</sup>	Average slope	-	2.5		mV/°C
$V_{25}^{(1)}$	Voltage at 25 °C	-	0.76		V
tSTART <sup>(2)</sup>	Startup time	-	6	10	μs
TS_temp <sup>(2)</sup>	ADC sampling time when reading the temperature (1 °C accuracy)	10	-	-	μs

# Display

- Emulate a display with *semi-hosting*
  - run with the debugger
  - Ada.Text\_IO output is redirected to the debugger



# Study the code and make changes

---

- All platforms:
  - Download zip file from moodle or  
[https://github.com/STR-UPM/OBDH\\_LABS](https://github.com/STR-UPM/OBDH_LABS)
  - Unzip the archive and move the resulting folder to your laboratory folder
  - Open project  
.../LAB3/Simple\_housekeeping/simple\_housekeeping.gpr
  - Build the executable and debug on the board

# Additional tasks

---

- Include the **conversion to Celsius in the Display.Put procedure.**
- Add the following statement to the main loop in the Housekeeping Run procedure:  
**delay until Clock + Milliseconds (1000);**  
(see the code in the Blinky\_LEDs program)

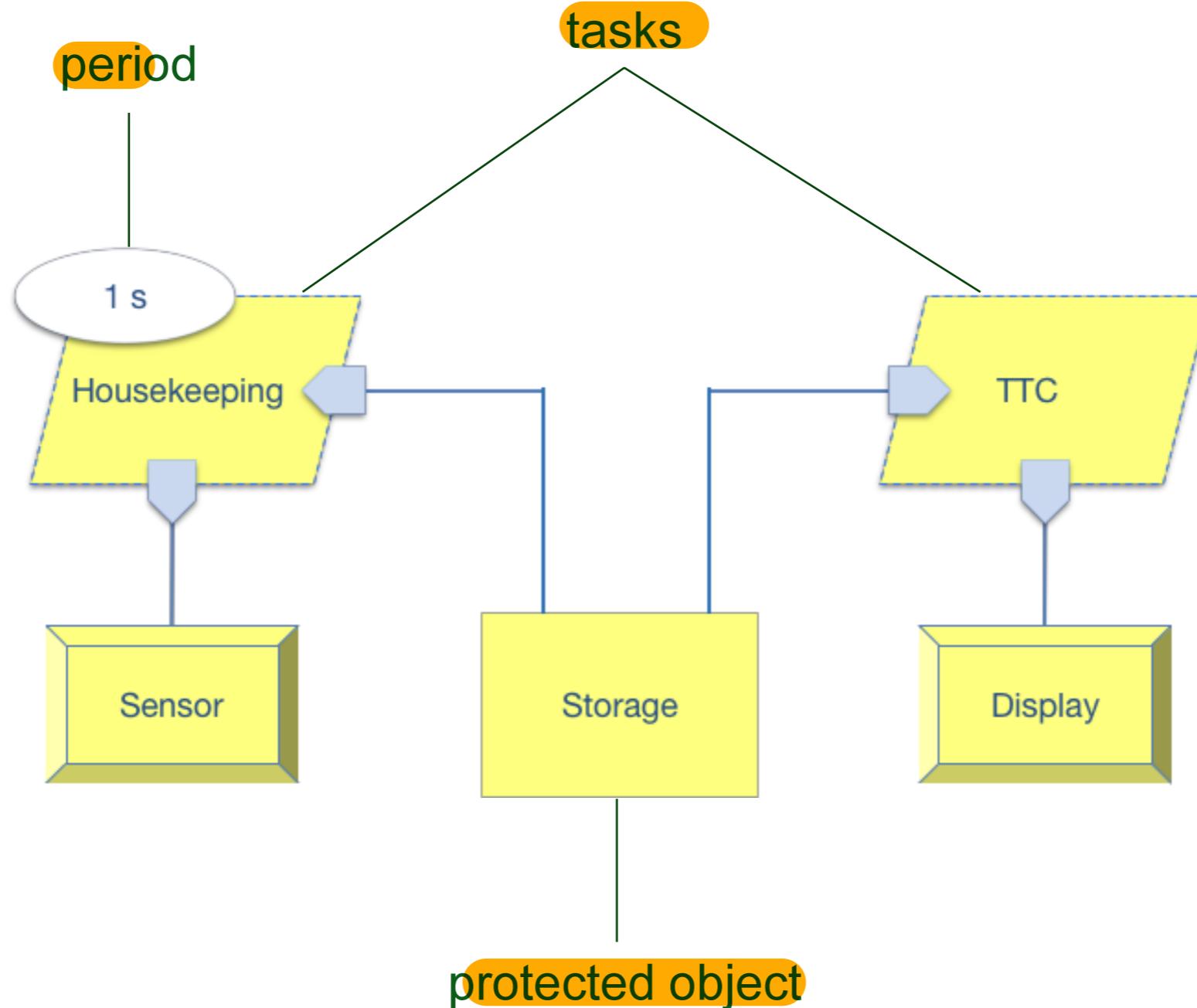
# 4. Tasking housekeeping system

---

- Add two tasks for better structuring
  - Housekeeping
    - read a sensor (OBC\_T)
  - TTC
    - display the sensor value
- Add a protected object for inter-task communication
  - Storage
    - the housekeeping task puts the sensor value in the storage
    - the TTC task gets the value from the storage

# Software architecture

---



# Study the code and make changes

---

- All platforms:
  - Download zip file from moodle or  
[https://github.com/STR-UPM/OBDH\\_LABS](https://github.com/STR-UPM/OBDH_LABS)
  - Unzip the archive and move the resulting folder to your laboratory folder
  - Open project  
  .../LAB4/Tasking\_housekeeping/tasking\_housekeeping.gpr
  - Build the executable and debug on the board

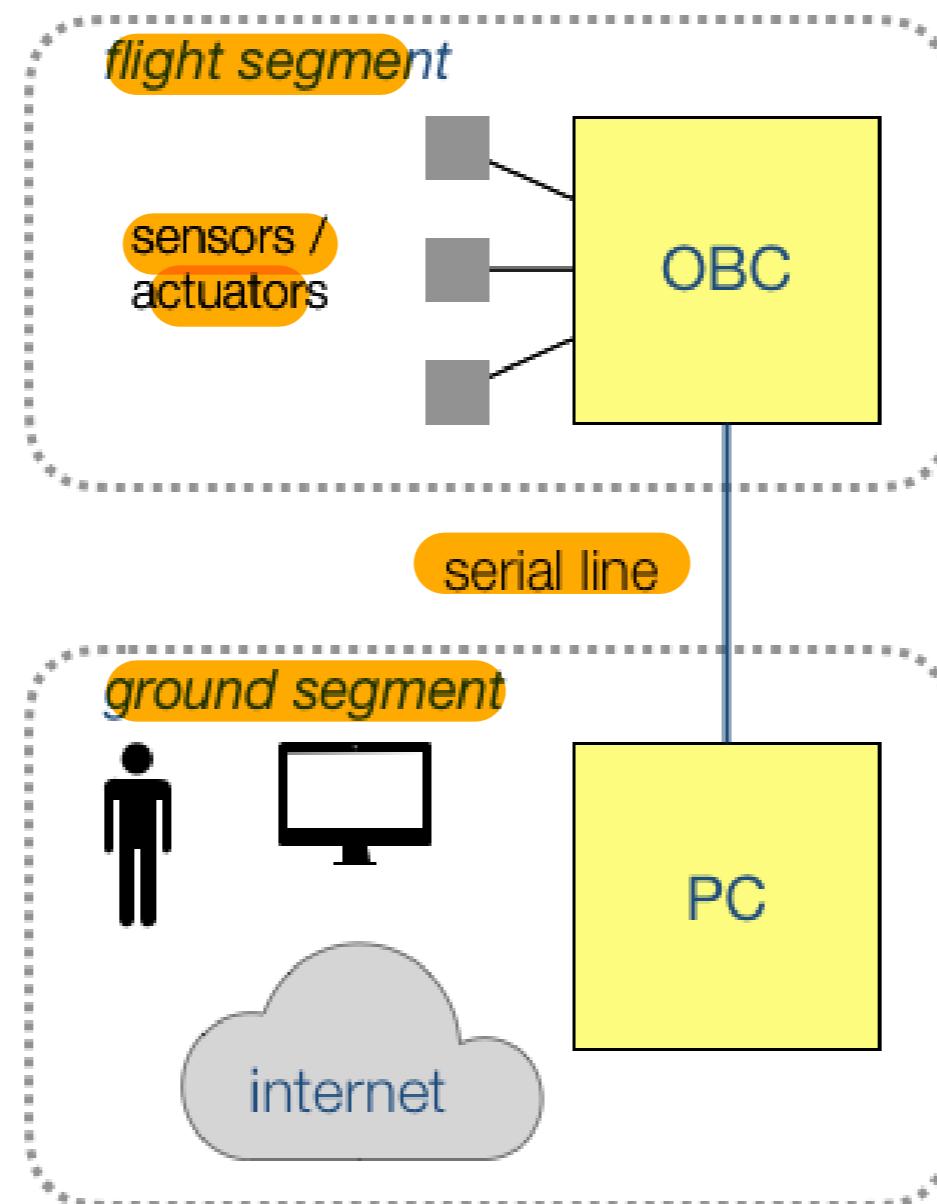
# Additional tasks

---

- Include the conversion to Celsius in the `Display.Put` procedure.
- Add the following statement to the main loop in the Housekeeping Run procedure:  
`delay until Clock + Milliseconds (1000);`  
(see the code in the `Blinky_LEDs` program)

# 5. Distributed housekeeping system

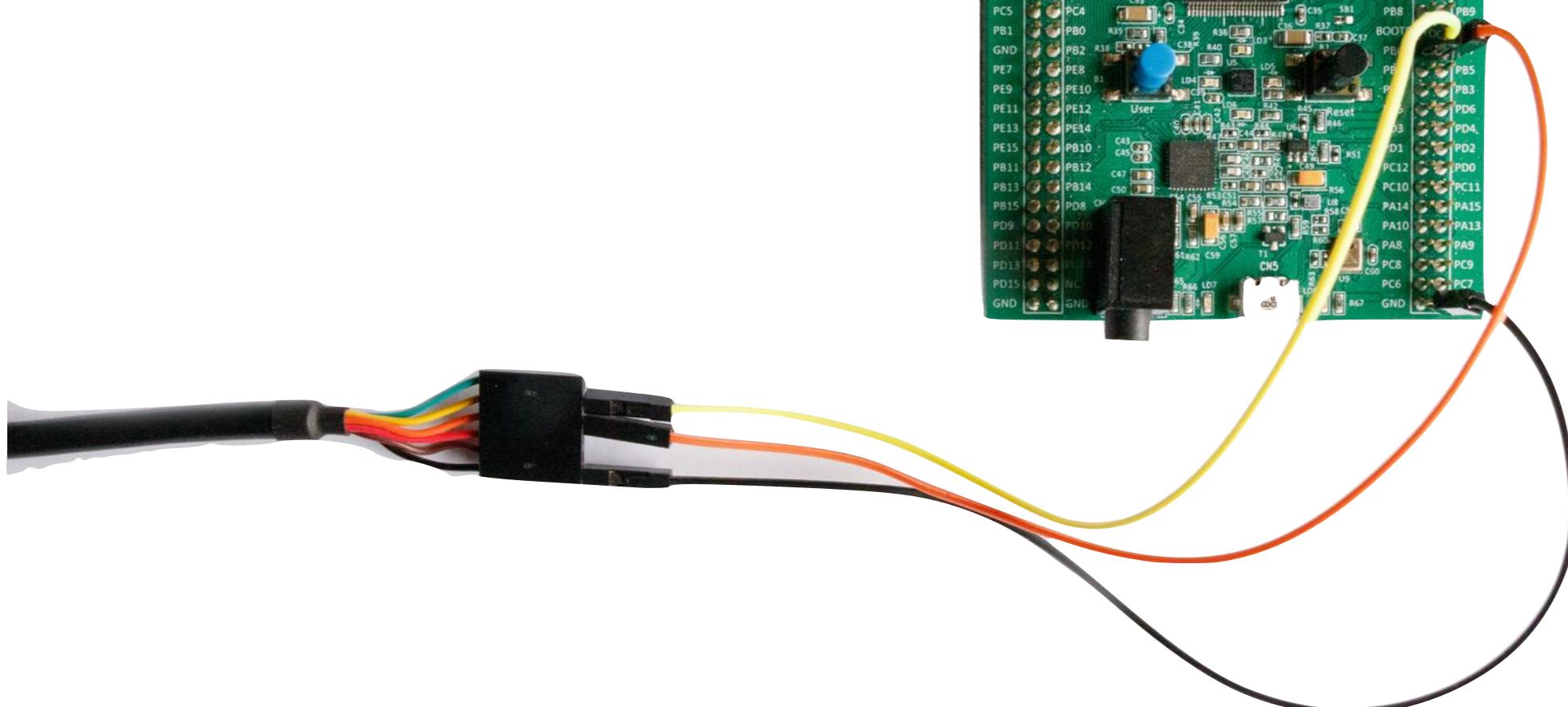
- Add a **serial interface connected to the host PC**
  - simulate a radio link between flight and ground systems



# Hardware

---

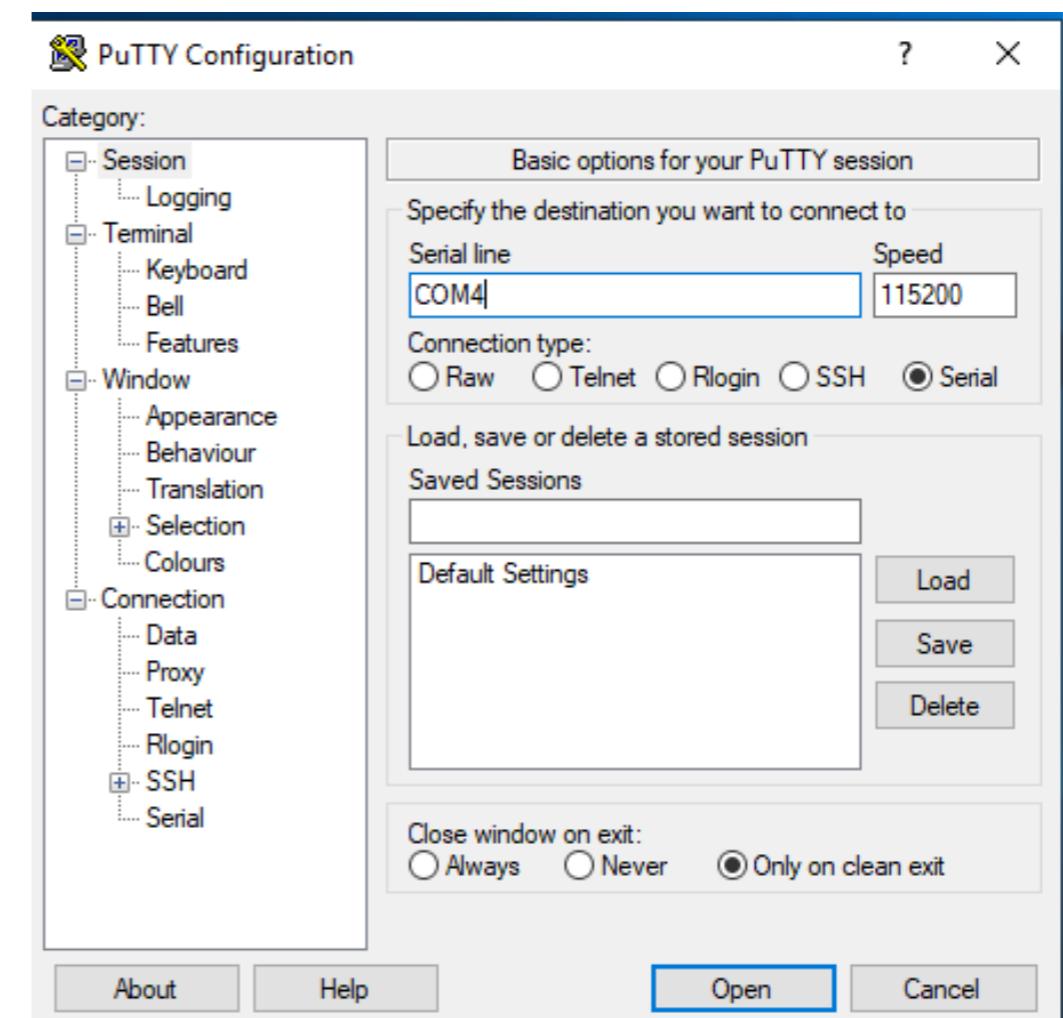
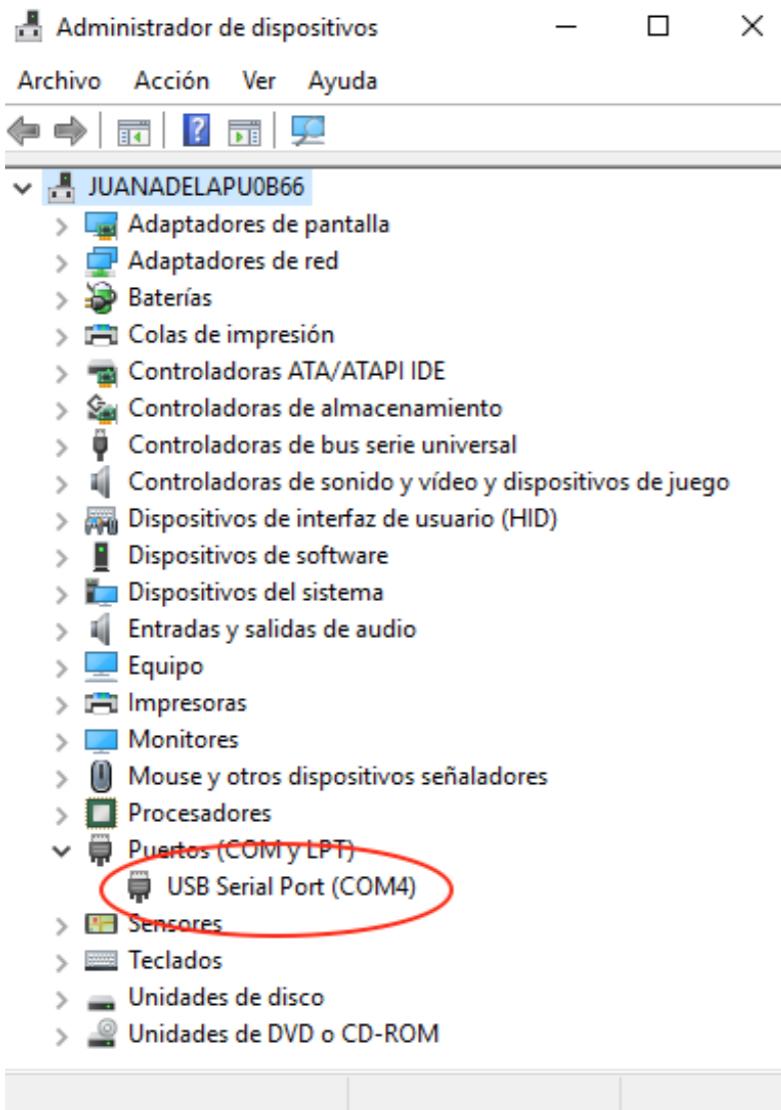
Connector pin	Board pin
1 (black)	GND
4 (orange)	PB7
5 (yellow)	PB6



# Host terminal

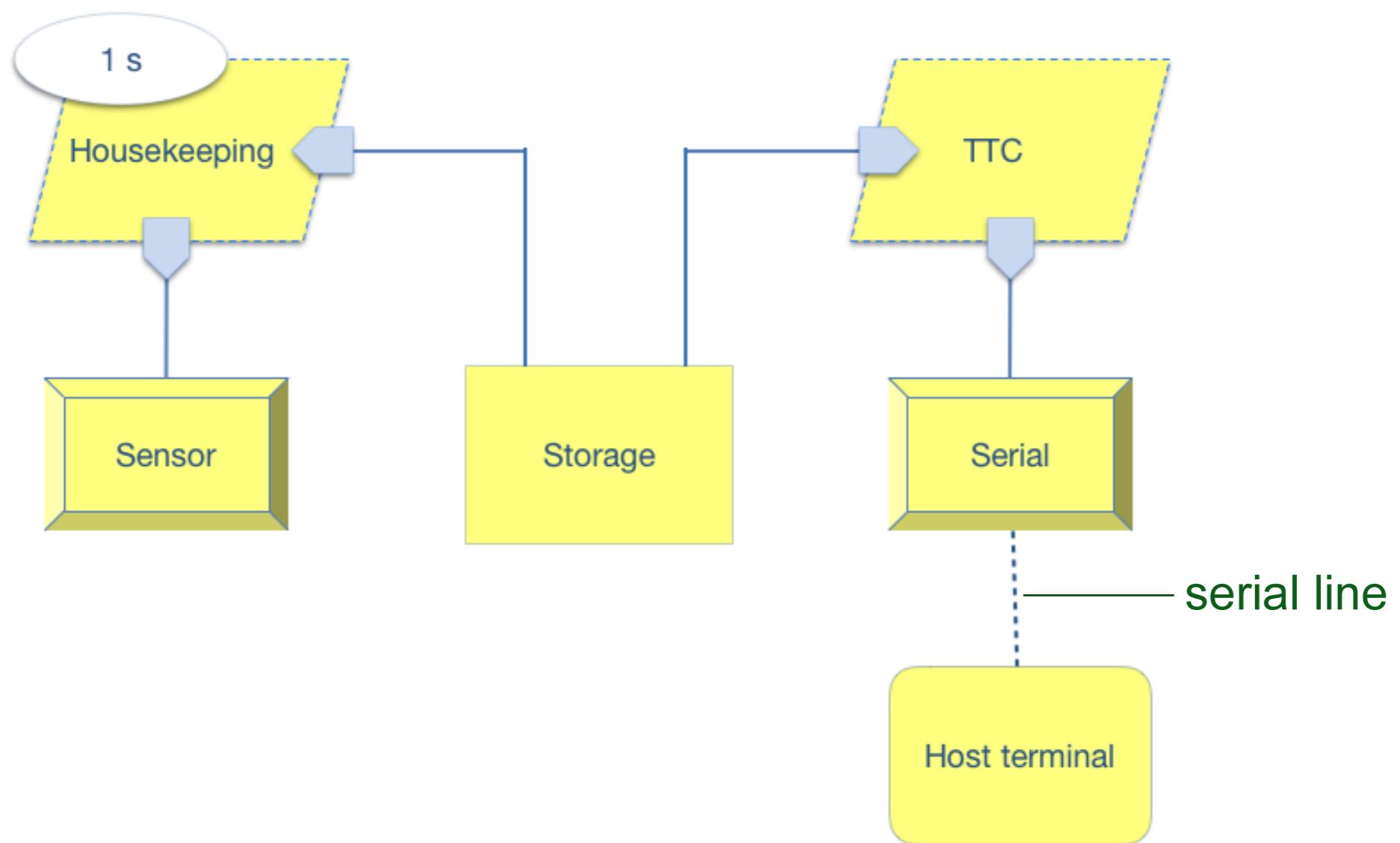
(Windows — for other platforms see manual)

- Download and install <https://www.putty.org>
- Find the COM port in device manager (e.g. COM4)
- Open PutTTY and configure as serial connection, 115200 bps, 8N1



# Software architecture

---



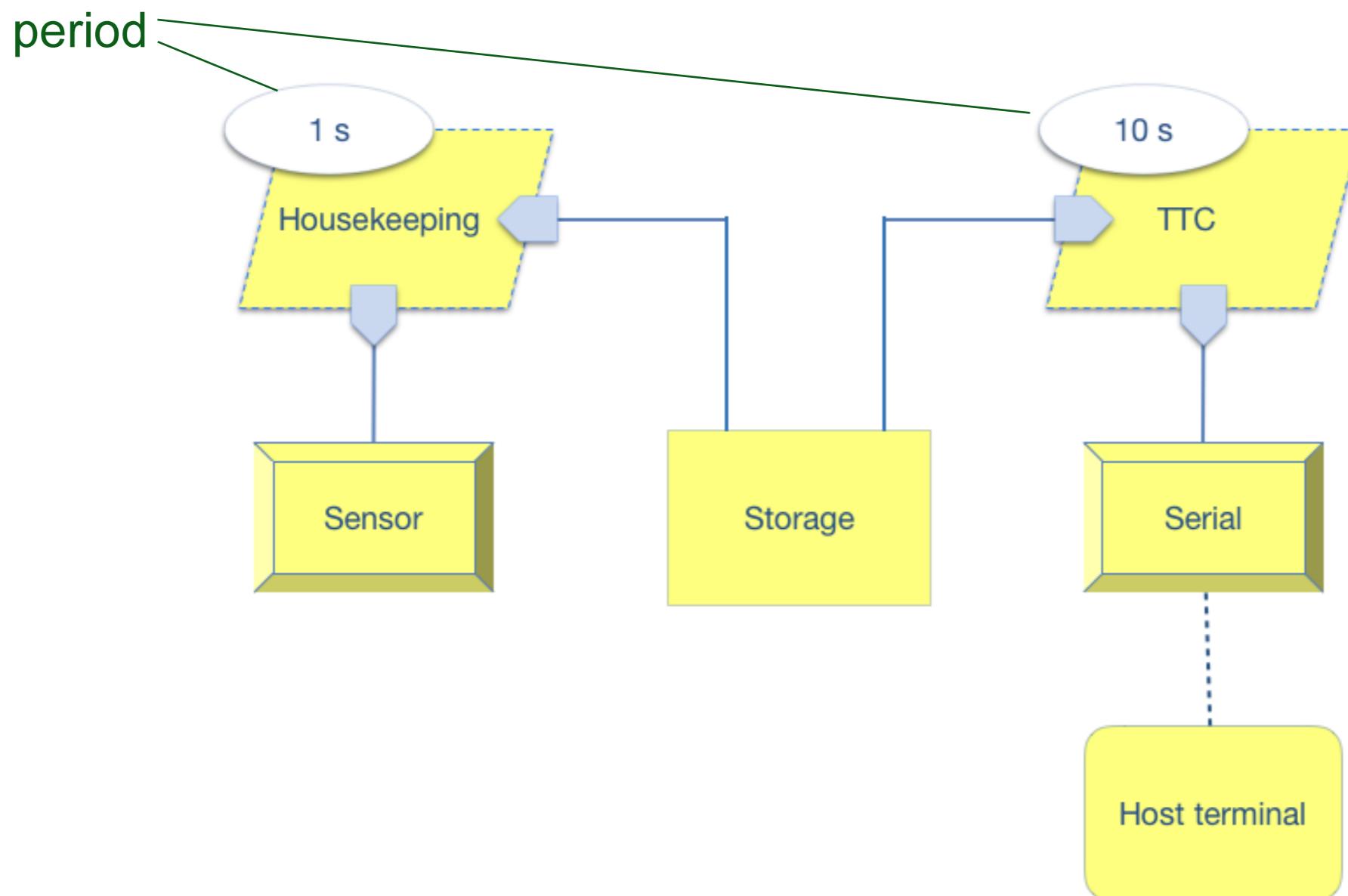
# Study the code and make changes

---

- All platforms:
  - Download zip file from moodle or  
[https://github.com/STR-UPM/OBDH\\_LABS](https://github.com/STR-UPM/OBDH_LABS)
  - Unzip the archive and move the resulting folder to your laboratory folder
  - Open project  
  `.../LAB5/Distributed_housekeeping/distributed_housekeeping.gpr`
  - Build the executable and flash to board

# 6. Real-time housekeeping system

- Add **real-time requirements and priorities** to tasks
- Add **timestamps to measurements**



# Real-time requirements

---

Task	Period	Deadline	Priority
Housekeeping	1,0	1,0	20
TTC	10,0	2,0	10
Storage Buffer	—	—	20

```
package Housekeeping is
    Period    : Time_Span := Milliseconds (1000); -- 1s
    HK_Priority : constant System.Priority := 20;
    ...
private
    Start_Delay : Time_Span := Milliseconds (1000); -- 1s
task Housekeeping_Task
    with Priority => HK_Priority;
end Housekeeping;
```

# Implementation

---

```
task body Housekeeping_Task is
    Data      : State;
    Next_Time : Time := Clock + Start_Delay;
begin
    loop
        delay until Next_Time;
        ...
        Next_Time := Next_Time + Period;
    end loop;
end Housekeeping_Task;
```

# Timestamps

---

```
package Housekeeping_Data is
    ...
    type Mission_Time is new Uint64;
    -- Seconds since system startup

    type State is
        record
            Data      : Analog_Data;
            Timestamp : Mission_Time;
        end record;

    end Housekeeping_Data;
```

# Housekeeping

---

```
procedure Get (S : out State) is
    OBC_T : Analog_Data;
    SC   : Seconds_Count;
    TS   : Time_Span;
begin
    Sensor.Get (OBC_T);
    Split (Clock, SC, TS);
    S.Data := OBC_T;
    S.Timestamp := Mission_Time (SC);
end Get;
```

# Storage

---

```
package Storage is

    procedure Put (Data : State);
    procedure Get (Data : out State);

    ...

    -- Real-time attributes
    ST_Priority : constant System.Priority := 20;

    private

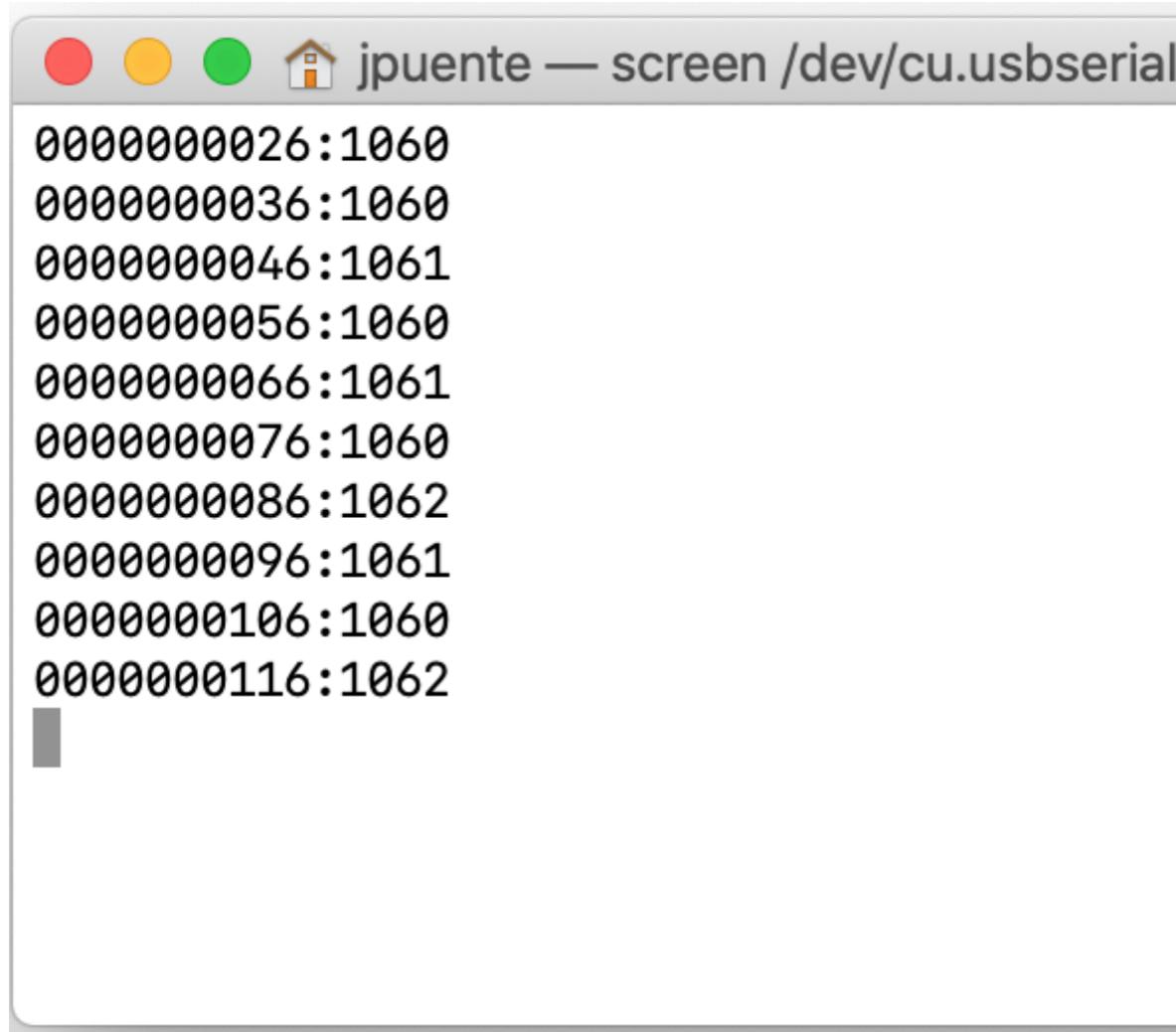
        protected Buffer
            with Priority => ST_Priority
        is ...

    ...

end Storage;
```

# Laboratory work

---

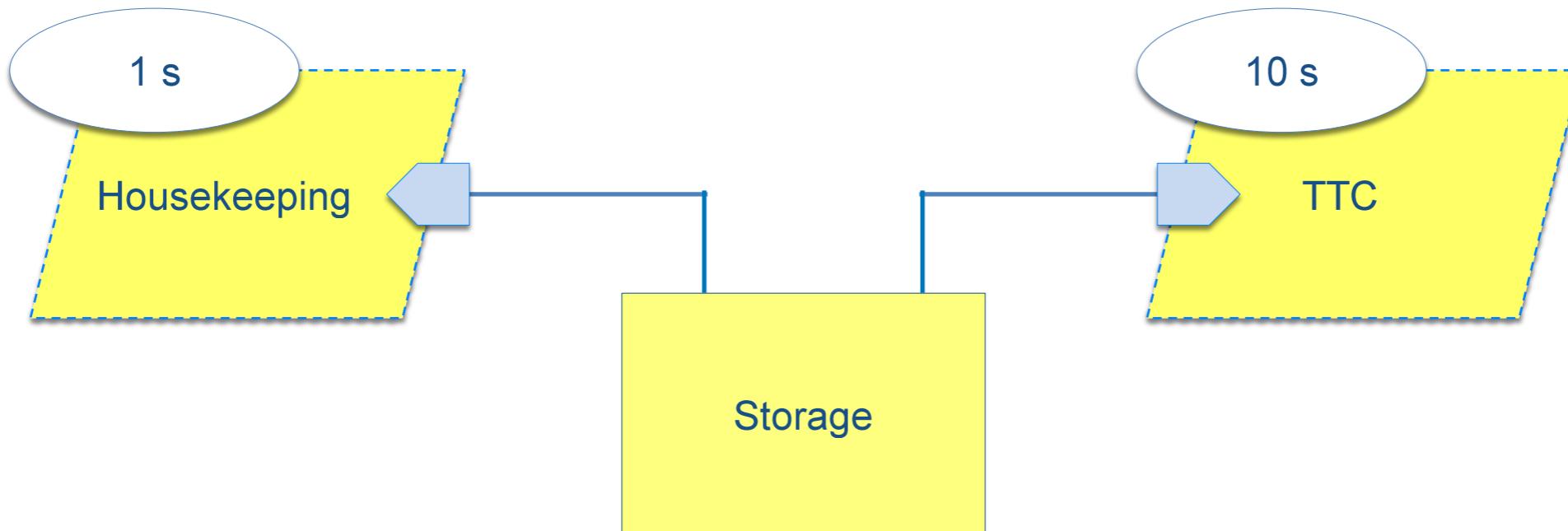
A screenshot of a terminal window titled "jpuente — screen /dev/cu.usbserial". The window shows a series of telemetry messages consisting of two numbers separated by a colon, such as "000000026:1060", "000000036:1060", etc., repeated ten times. A vertical scrollbar is visible on the right side of the terminal window.

```
000000026:1060
000000036:1060
000000046:1061
000000056:1060
000000066:1061
000000076:1060
000000086:1062
000000096:1061
000000106:1060
000000116:1062
```

- Open project  
.../LAB6/RT\_housekeeping/real\_time\_housekeeping.gpr
- Build the executable and flash to board
- Make sure the TTY cable is connected
- Telemetry messages appear on the ground terminal

# Real-time analysis

- Task structure



- Real-time attributes

Task	T	C	B	D	R	P	$C_{Put}$	$C_{Get}$
HK	1,0			1,0		20		
TTC	10,0			2,0		10		
						CP	20	20

# WCET measurement

---

- A simple dynamic measurement technique will be used in the laboratory
- Execute task bodies a number of times and measure times with the board real-time clock
  - e.g. for HK task

```
T1 := Clock;  
for I in 1..N loop  
    Get(Data);  
    Storage.Put (Data);  
    T := T + Period;  
end loop;  
T2 := Clock;  
C := T2 - T1;
```

- the estimated WCET in seconds is C/N

# Laboratory work

---

```
● ● ● jpuente — screen /dev/cu.usbserial-FTA5I2
Start test no 1
HK ( 1000000 times) : 13.027373679 s
TC ( 1000000 times) : 26.069529321 s
ST
Put ( 1000000 times) : 2.561030637 s
Get ( 1000000 times) : 3.448276304 s
```

- Open project  
.../LAB6/RT\_housekeeping/wcet\_meter.gpr
- Study the source code
- Build the executable and flash to board
- Make sure the TTY cable is connected
- Analysis results appear on the ground terminal

# Real-time analysis with measured values

---

Task	T	C	B	D	R	P	$C_{Put}$	$C_{Get}$
HK	1,0	$13 \cdot 10^{-6}$	$4 \cdot 10^{-6}$	1,0		20	$3 \cdot 10^{-6}$	—
TTC	10,0	$26 \cdot 10^{-6}$	—	2,0		10	—	$4 \cdot 10^{-6}$
						CP	20	20

$$R_i = C_i + B_i + \sum_{j \in \text{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Task	T	C	B	D	R	P	$C_{Put}$	$C_{Get}$
HK	1,0	$13 \cdot 10^{-6}$	$4 \cdot 10^{-6}$	1,0	$17 \cdot 10^{-6}$	20	$3 \cdot 10^{-6}$	—
TTC	10,0	$26 \cdot 10^{-6}$	—	2,0	$39 \cdot 10^{-6}$	10	—	$4 \cdot 10^{-6}$
						CP	20	20

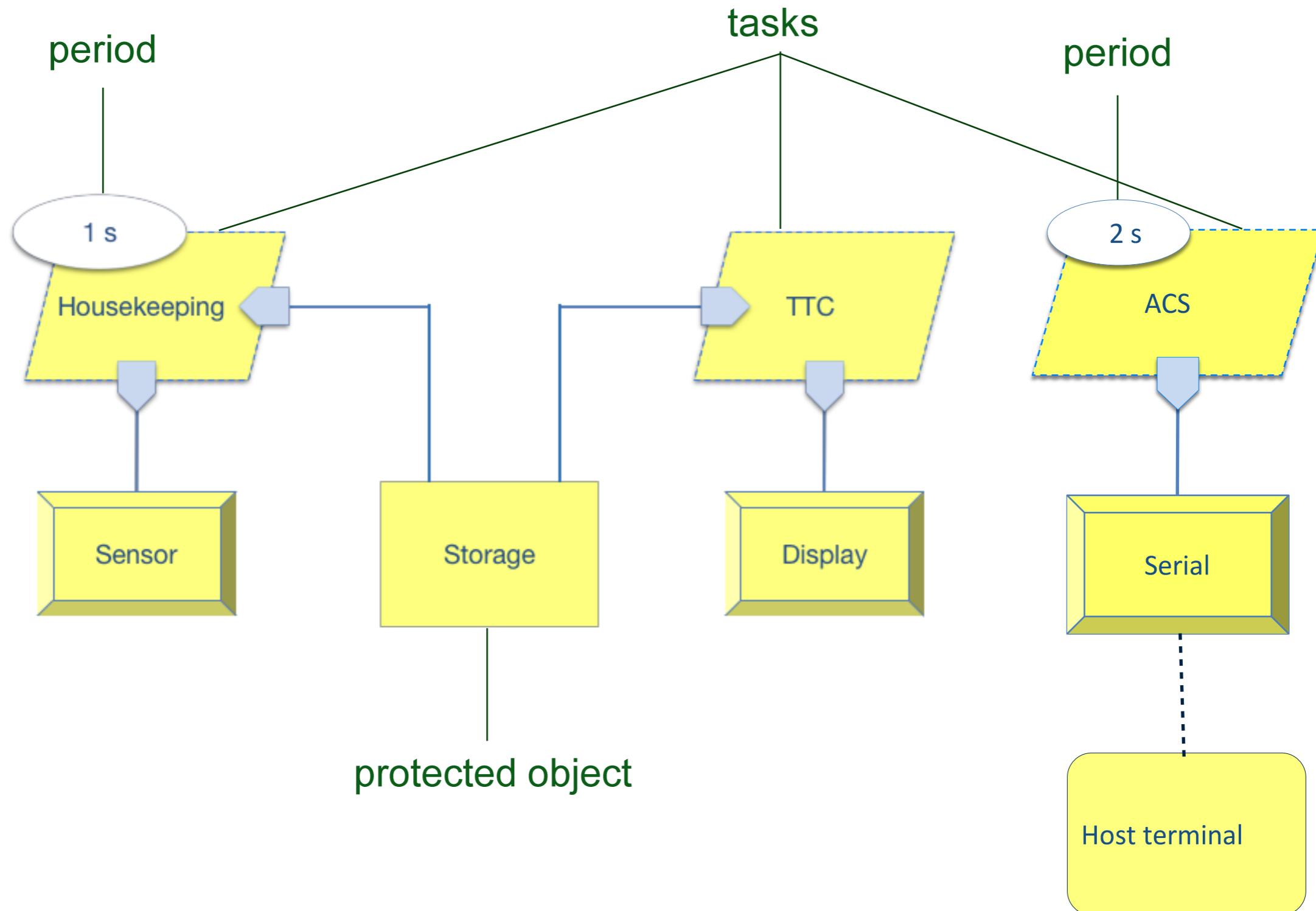
All deadlines are guaranteed

# 7. Tasking housekeeping and ACS system

---

- **Include three tasks:**
  - **Housekeeping**
    - read a sensor (OBC\_T)
  - **TTC**
    - display the sensor value using semi-hosting
  - **ACS**
    - control the attitude of the satellite
- Host computer runs Simulink model of the environment
  - Host and target interchange sensor readings and actuator outputs by serial line

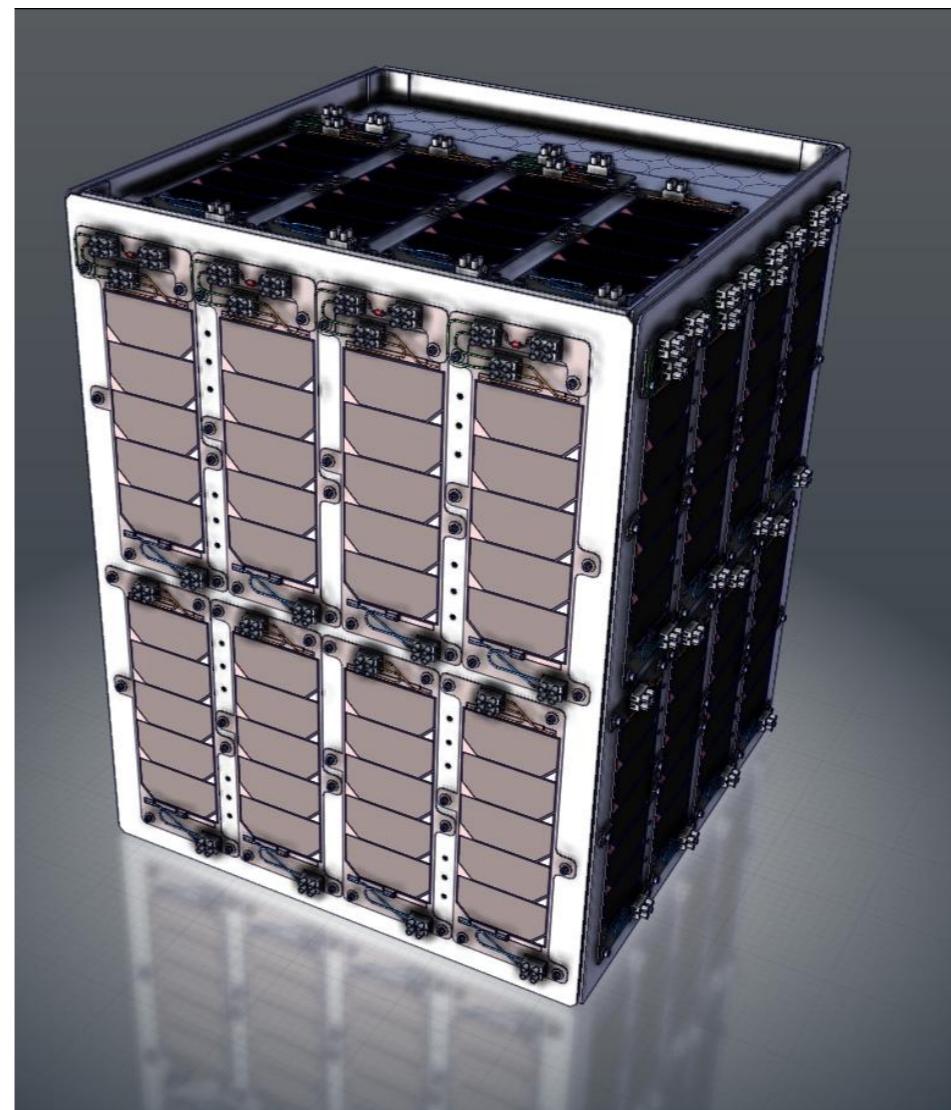
# Software architecture



# UPMSat-2

---

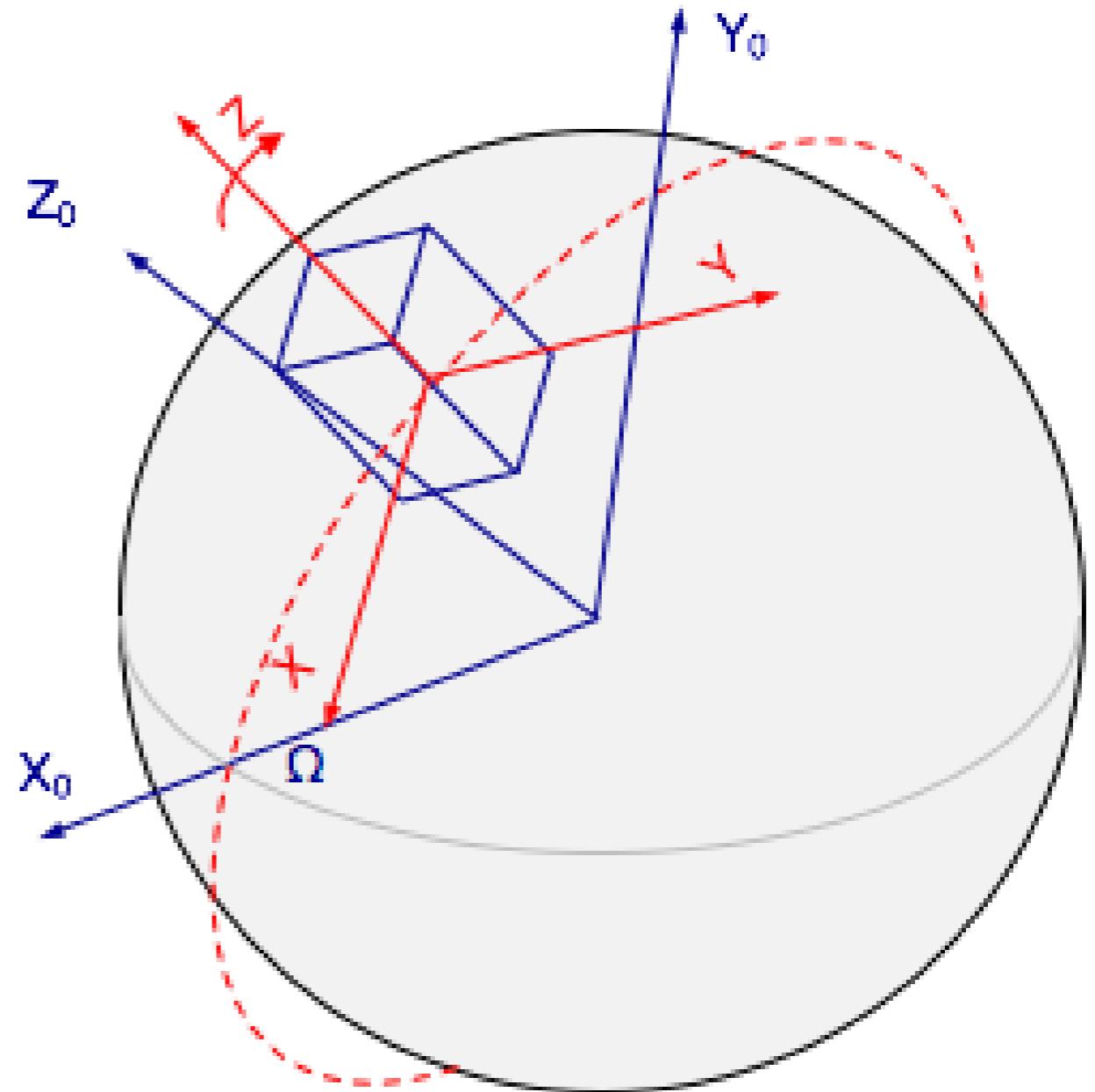
- Experimental micro-satellite
  - $0.50 \times 0.50 \times 0.60$  m, 50 kg
  - sun-synchronous orbit
    - 700 km, 98 min period
  - powered by solar panels
  - VHF radio dual link 400 MHz
- Dipole antenna on top face
  - must be visible from Earth at all times



# Attitude control frames

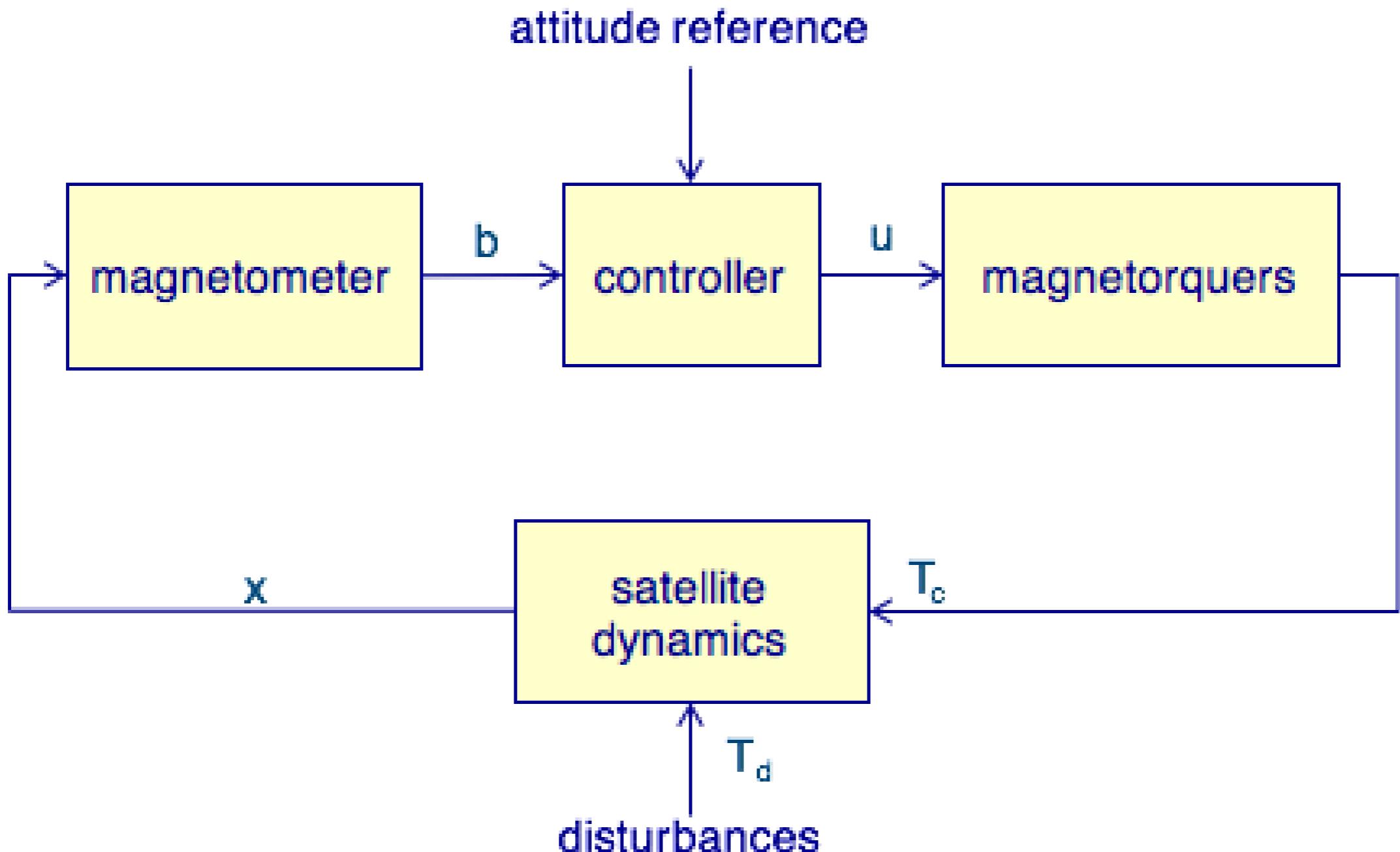
---

- Body reference frame XYZ
  - antenna on Z axis
- Orbital reference frame  $X_0Y_0Z_0$ 
  - inertial for short intervals
- Attitude reference
  - align Z with  $Z_0$
  - slow rotation around Z



# Magnetic attitude control

---



# Magnetometers / magnetorquers

---

- Magnetometers provide a measurement of the Earth's magnetic field components in the body frame
  - $\mathbf{b}$  : magnetic field vector
- Magnetorquers generate a control torque normal to the orbit
  - $\mathbf{m}$ : magnetic moment vector
    - ✓ proportional to the current in each magnetorquer coil
    - $m_i = AN \cdot I_i$

# Validation steps: Model-in-the-loop

---

- Preliminary validation of control algorithm
- Use a model of the ACS, together with models of the space environment and the spacecraft dynamics, to assess the validity of the control law and the design parameters.
- Usually carried out by a control engineer using a simulation tool.
  - Simulink is commonly used for ACS development.

# Validation steps: Processor-in-the-loop

---

- A further step to full validation is running the control software on actual hardware, while still using simulation models of the sensors, actuators, and spacecraft dynamics.
- A software validation facility (SVF) uses an auxiliary computer, linked to the OBC by a serial line, to run a simulation model of the Earth's magnetic field and the satellite dynamics.
  - In this way, engineering values (Nm and T) can be interchanged.
- The code of the control algorithm is generated with the Simulink Code Generator, and compiled and linked with the on-board software using the GNAT development environment.
- Functional & real-time behaviour can be validated.

# Validation steps: **Hardware-in-the-loop**

---

- The next step is to include a model of the actual magnetometers and magnetorquers in the simulation model, and connect their inputs and outputs to the OBC hardware by means of its analog and digital input and output lines.
- This simple kind of HIL setup allows the operation of input/output devices, as well as signal conditioning hardware and software, to be validated.
- Full HIL validation, though, requires using the real magnetometers and magnetorquers hardware and, ultimately, the whole satellite structure, on a setup which enables the attitude of the spacecraft to evolve in a free fashion.

# Technology Readiness Level (TRL)

---

The technical maturity of instruments and spacecraft sub-systems with respect to a specific space application are classified according to a "Technology Readiness Level" (TRL) on a scale of 1 to 9. ESA uses the ISO standard 16290 Space systems – Definition of the Technology Readiness Levels (TRLs) and their criteria assessment.

## ISO Technology Readiness Level Summary

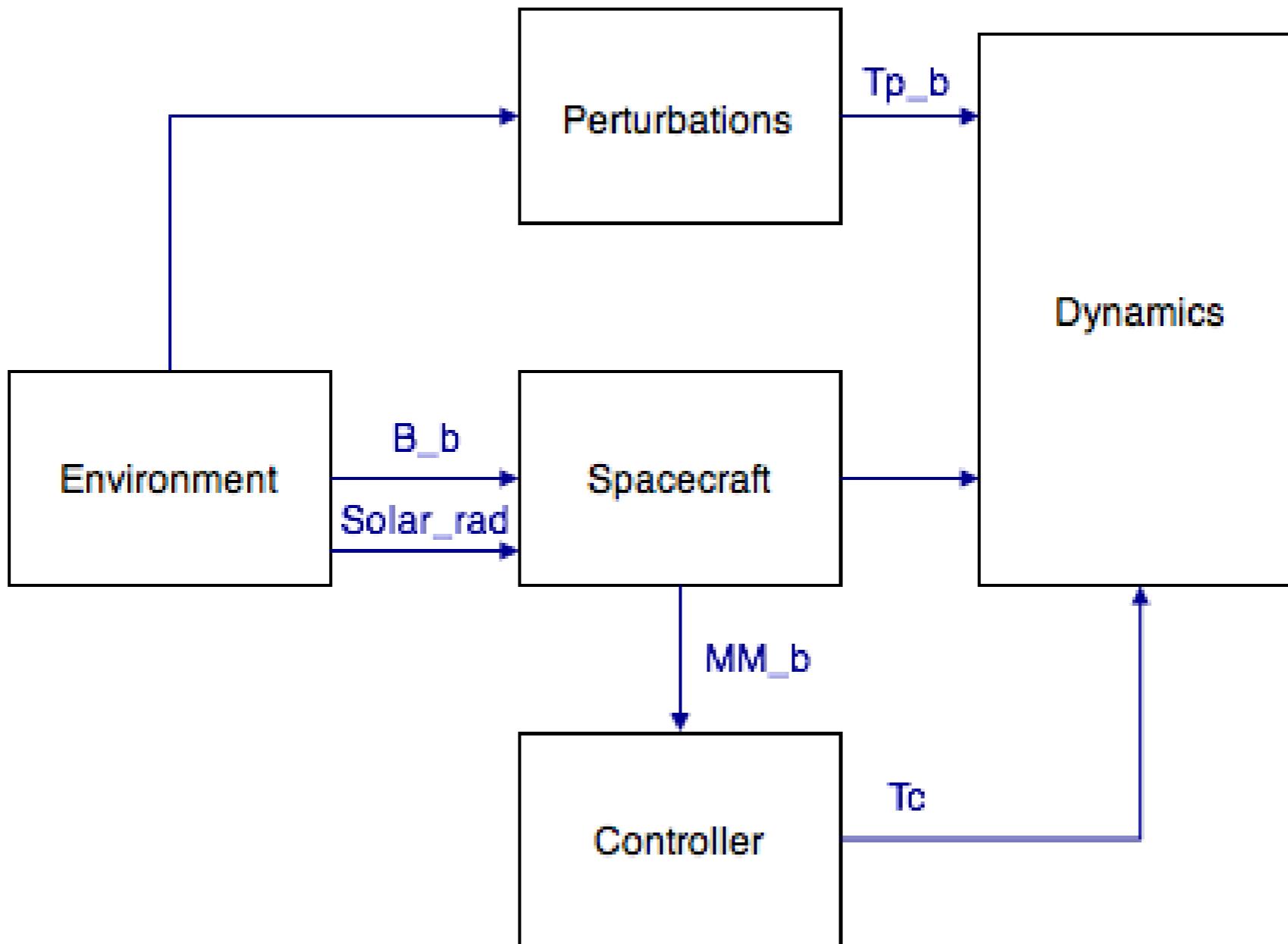
### TRL Level Description

- 1 Basic principles observed and reported
- 2 Technology concept and/or application formulated
- 3 Analytical and experimental critical function and/or characteristic proof-of-concept
- 4 Component and/or breadboard functional verification in laboratory environment
- 5 Component and/or breadboard critical function verification in relevant environment
- 6 Model demonstrating the critical functions of the element in a relevant environment
- 7 Model demonstrating the element performance for the operational environment
- 8 Actual system completed and accepted for flight ("flight qualified")
- 9 Actual system "flight proven" through successful mission operations

<http://sci.esa.int/sci-ft/50124-technology-readiness-level>

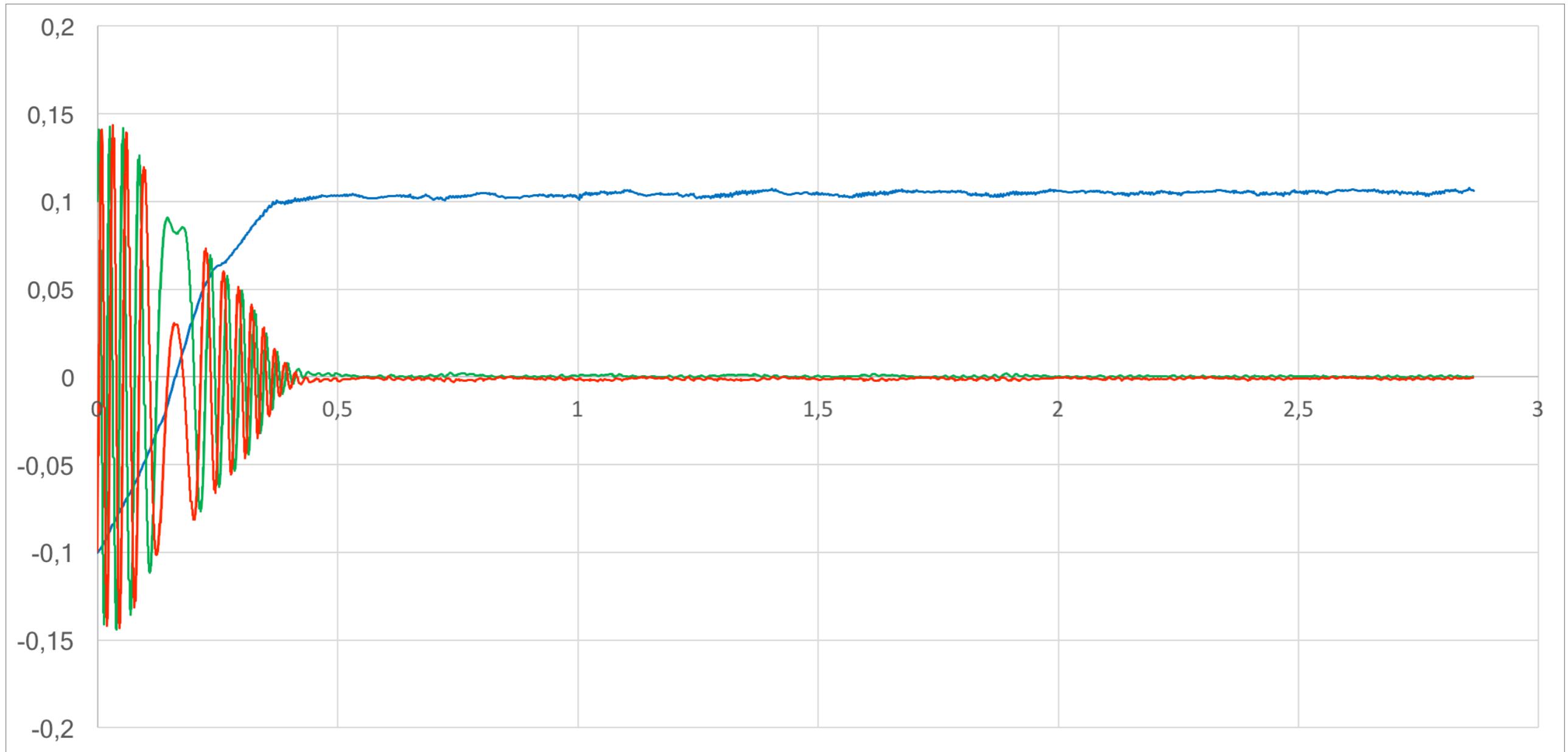
# ACS MIL (Model In the Loop) validation

---



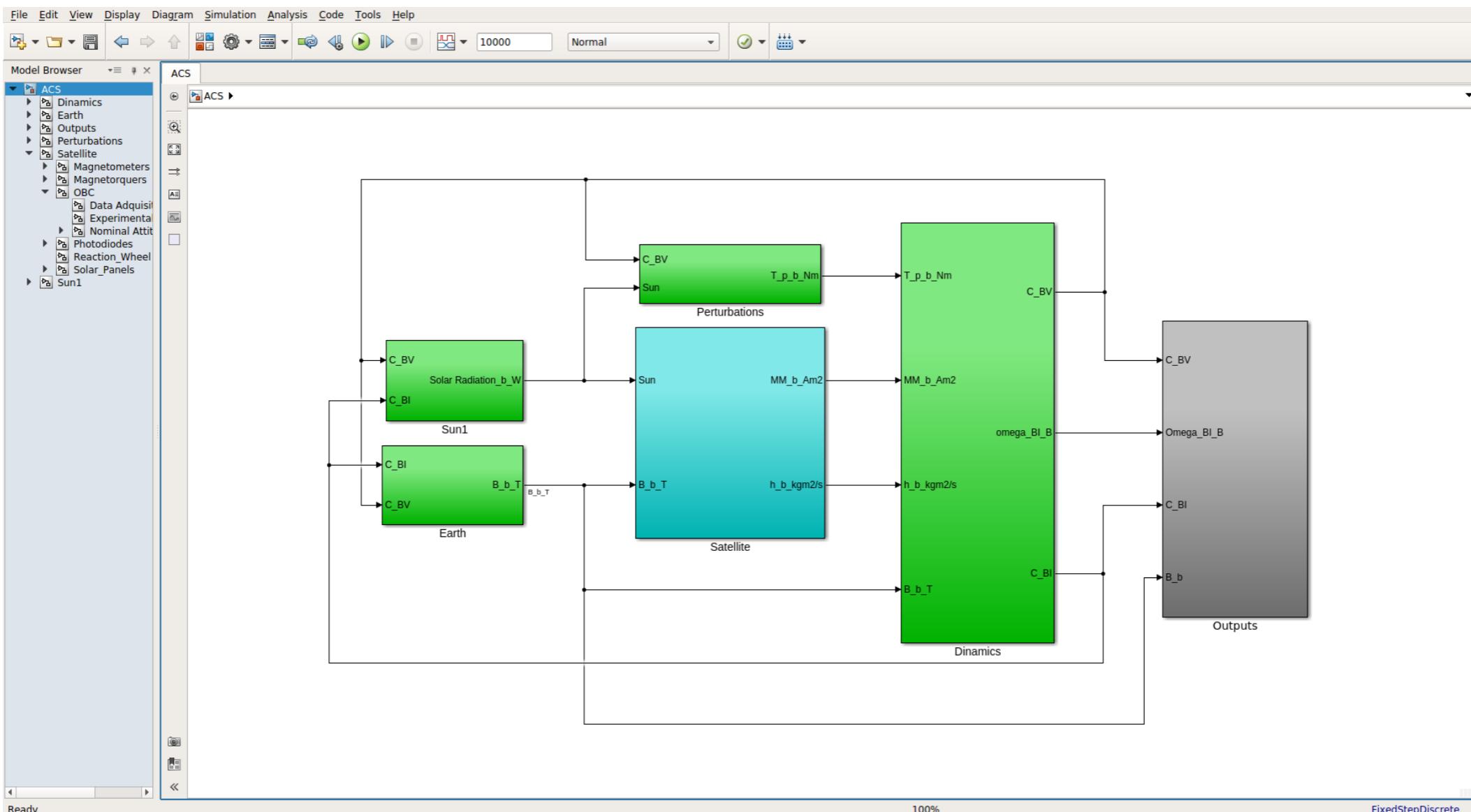
# Angular velocity stabilization

---



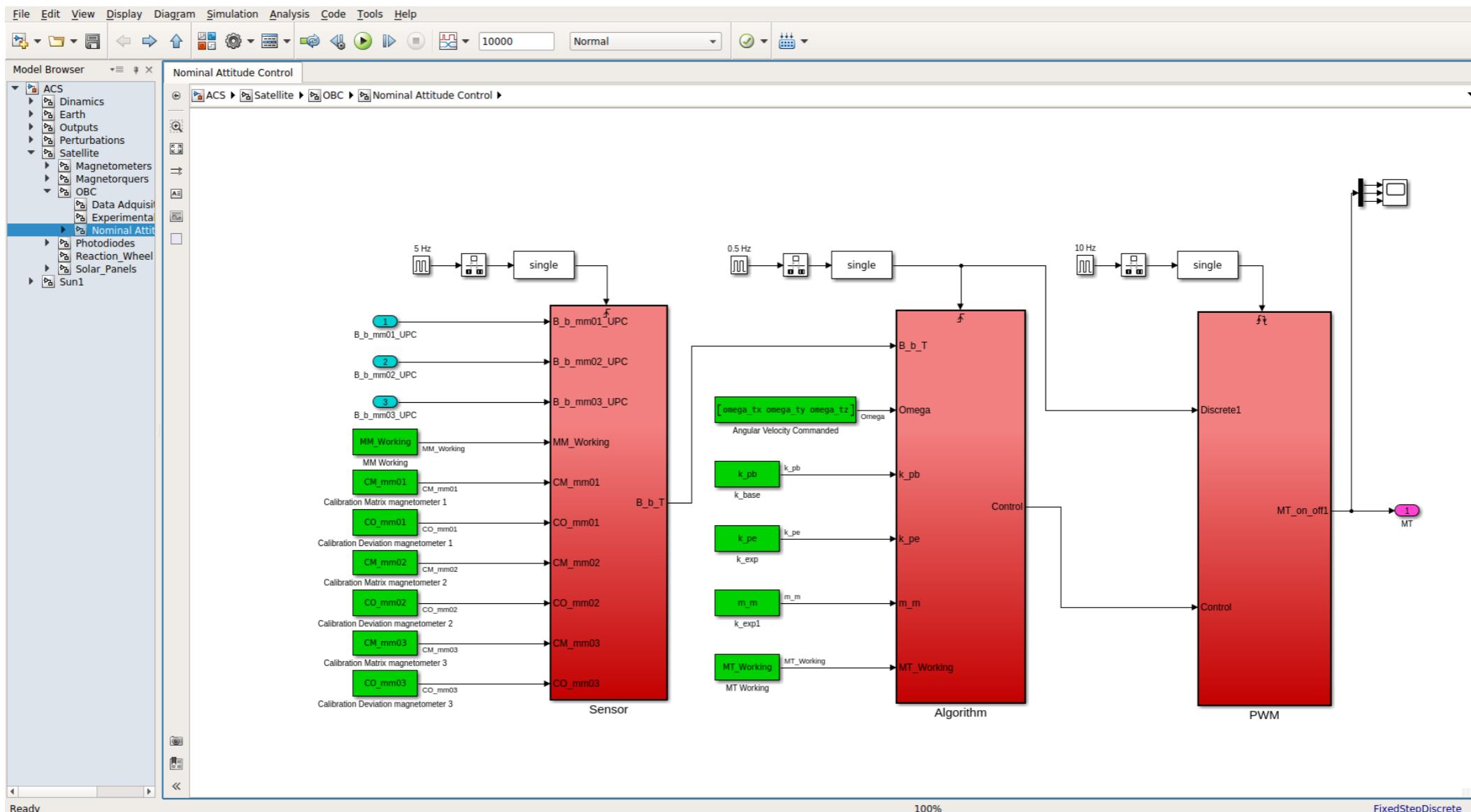
# Simulink model

- Run matlab and open ACS.slx from directory LAB/ACS.



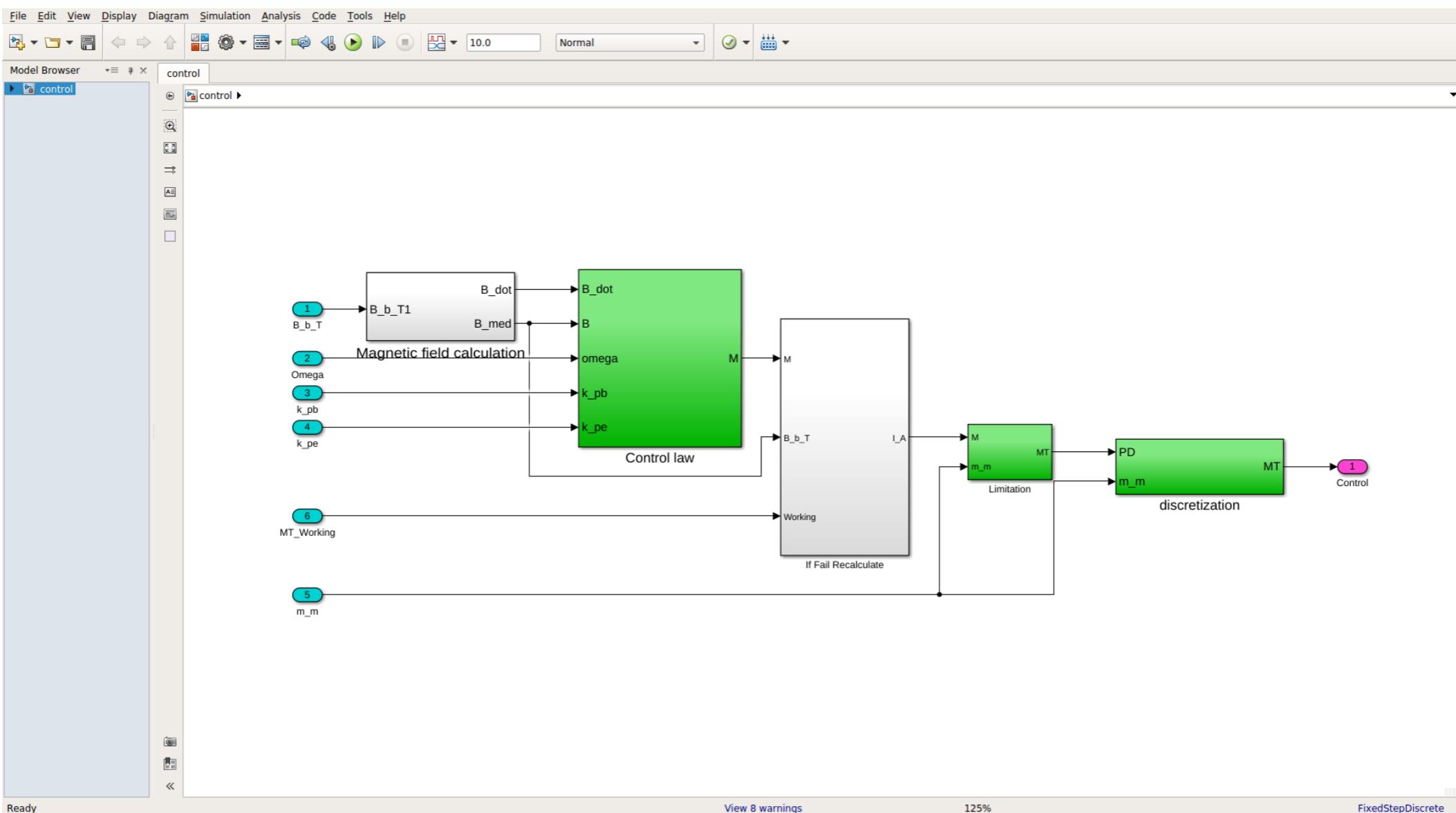
# Simulink model

- Navigate to nominal attitude control.
- Simulate and verify angular velocity stabilization



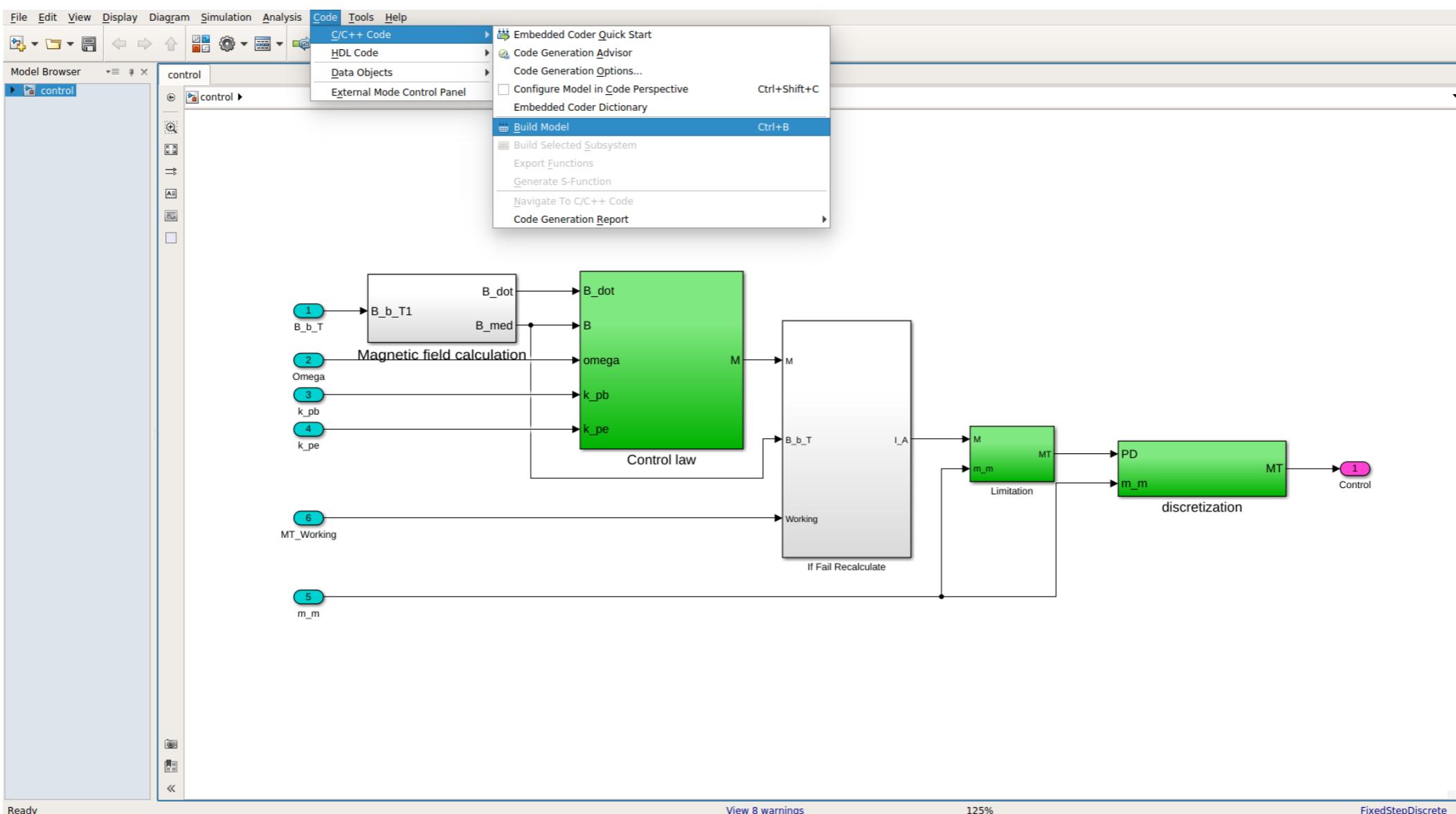
# Simulink model for code generation

- Open control.slx from directory LAB/ACS\_PIL



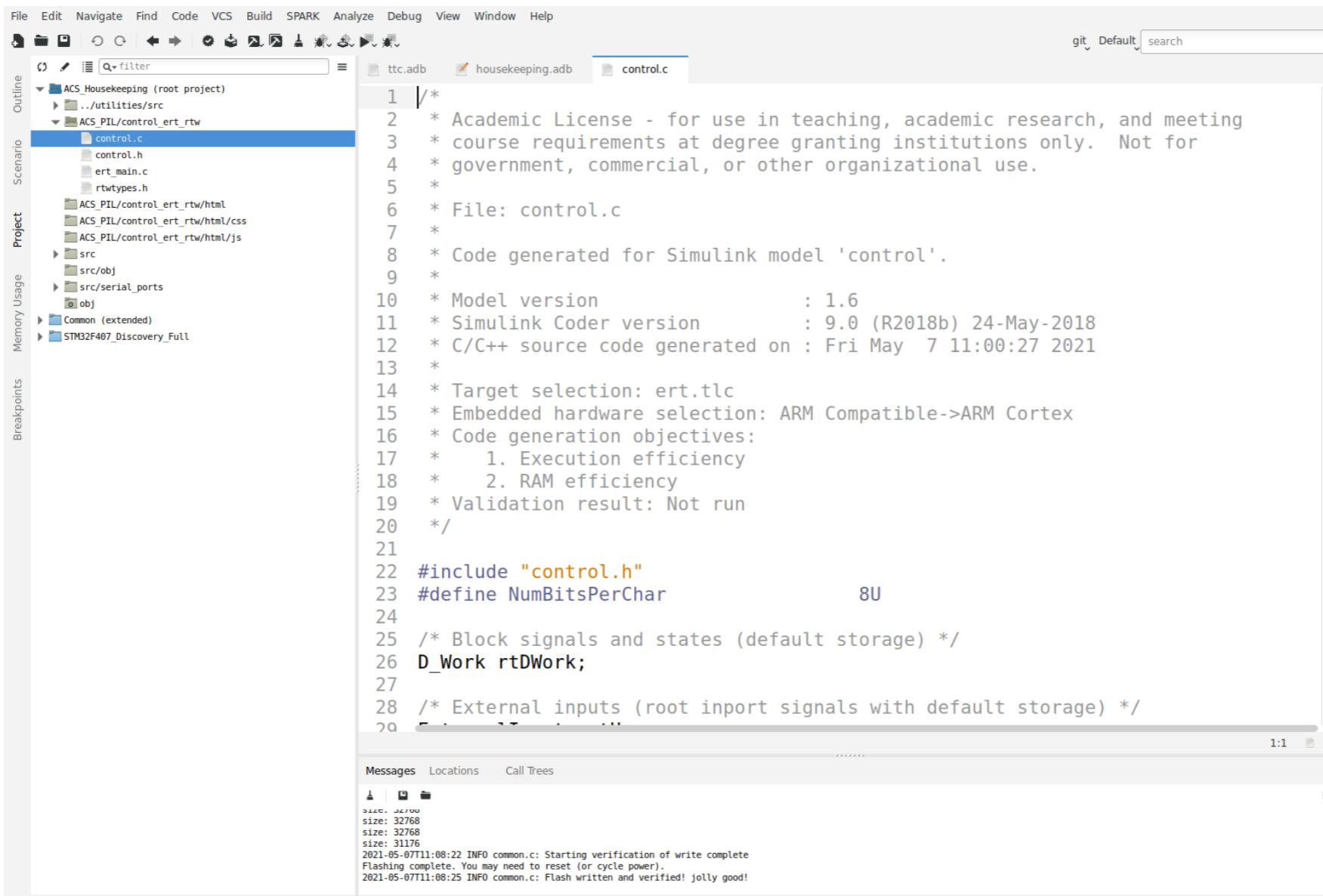
# Code generation

- Generate code with Embedded Coder toolbox
- Take a look to code generation report



# Code generation

- The generated C code is located at **LAB7/ACS\_PIL/control\_ert\_rtw**
- Launch gnatstudio and open **LAB7/ACS\_housekeeping.gpr**



The screenshot shows the gnatstudio IDE interface. The left sidebar displays the project structure under the 'Project' tab, showing a root project 'ACS\_Housekeeping' with subfolders like 'ACS\_PIL/control\_ert\_rtw' which contains 'control.c'. The main editor window shows the content of 'control.c'. The code is a generated C file for a Simulink model 'control'. It includes a multi-line comment at the top detailing the academic license, the model version (1.6), the Simulink Coder version (9.0 R2018b from May 24, 2018), and the date of generation (Fri May 7 11:00:27 2021). The code then defines '#include "control.h"', '#define NumBitsPerChar 8U', and declares 'D\_Work rtDWork;'. Below the code, the status bar shows messages related to file sizes and verification logs.

```
/*
 * Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only. Not for
 * government, commercial, or other organizational use.
 *
 * File: control.c
 *
 * Code generated for Simulink model 'control'.
 *
 * Model version : 1.6
 * Simulink Coder version : 9.0 (R2018b) 24-May-2018
 * C/C++ source code generated on : Fri May 7 11:00:27 2021
 *
 * Target selection: ert.tlc
 * Embedded hardware selection: ARM Compatible->ARM Cortex
 * Code generation objectives:
 *   1. Execution efficiency
 *   2. RAM efficiency
 * Validation result: Not run
 */

#include "control.h"
#define NumBitsPerChar 8U

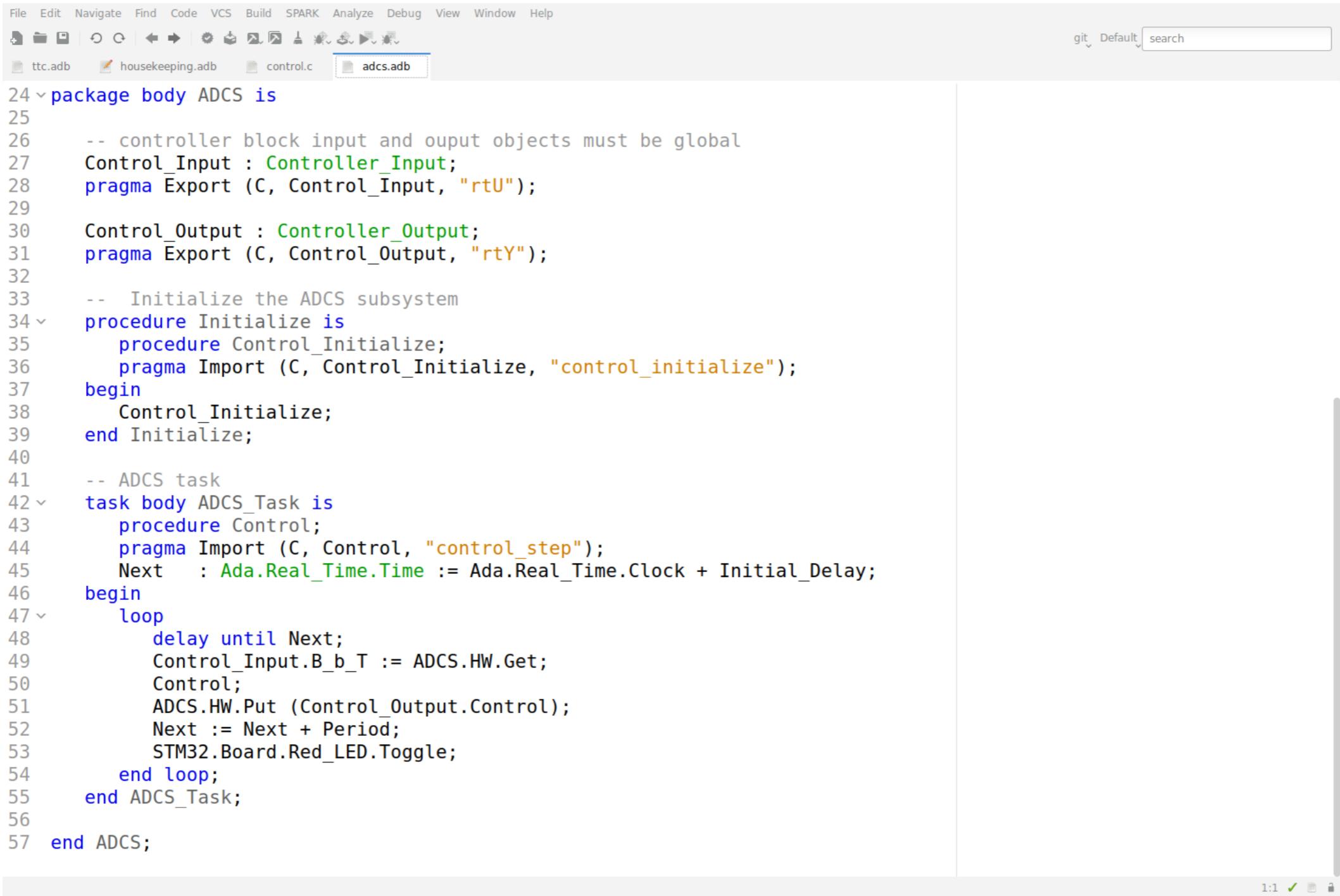
/* Block signals and states (default storage) */
D_Work rtDWork;

/* External inputs (root import signals with default storage) */

size: 32768
size: 32768
size: 31176
2021-05-07T11:08:22 INFO common.c: Starting verification of write complete
Flashing complete. You may need to reset (or cycle power).
2021-05-07T11:08:25 INFO common.c: Flash written and verified! jolly good!
```

# ACS implementation

---



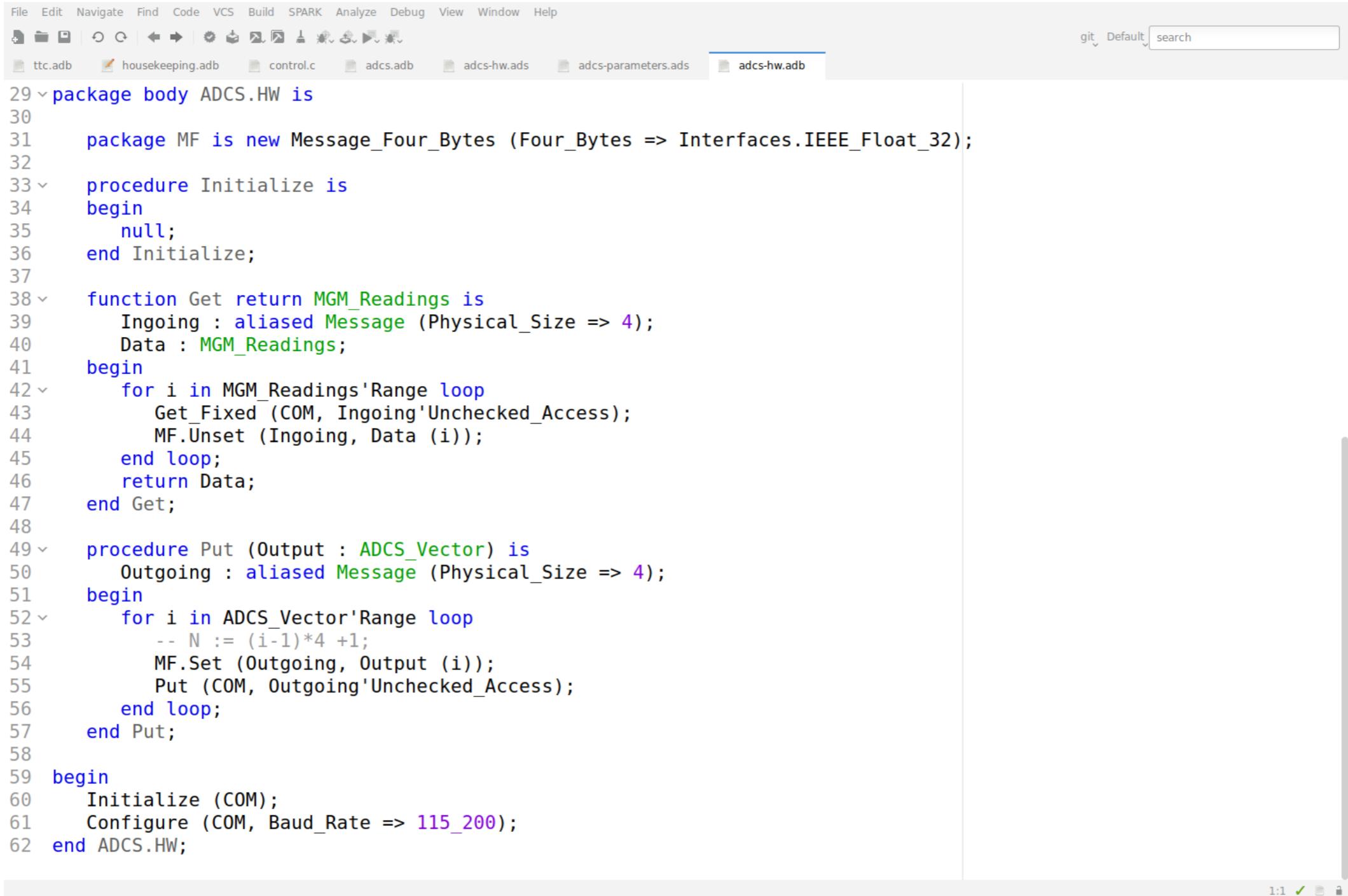
The screenshot shows a code editor window with the following details:

- Toolbar:** File, Edit, Navigate, Find, Code, VCS, Build, SPARK, Analyze, Debug, View, Window, Help.
- Search Bar:** git Default search
- File List:** ttc.adb, housekeeping.adb, control.c, adcs.adb (the active file)
- Code Content:** Ada source code for the ADCS package body. The code defines a package body named ADCS with various subprograms and types. It includes imports from C and Ada, and uses Ada Real-Time types like Time and Clock.

```
File Edit Navigate Find Code VCS Build SPARK Analyze Debug View Window Help
git Default search
ttc.adb housekeeping.adb control.c adcs.adb
24 package body ADCS is
25
26     -- controller block input and ouput objects must be global
27     Control_Input : Controller_Input;
28     pragma Export (C, Control_Input, "rtU");
29
30     Control_Output : Controller_Output;
31     pragma Export (C, Control_Output, "rtY");
32
33     -- Initialize the ADCS subsystem
34     procedure Initialize is
35         procedure Control_Initialize;
36         pragma Import (C, Control_Initialize, "control_initialize");
37     begin
38         Control_Initialize;
39     end Initialize;
40
41     -- ADCS task
42     task body ADCS_Task is
43         procedure Control;
44         pragma Import (C, Control, "control_step");
45         Next    : Ada.Real_Time.Time := Ada.Real_Time.Clock + Initial_Delay;
46     begin
47         loop
48             delay until Next;
49             Control_Input.B_b_T := ADCS.HW.Get;
50             Control;
51             ADCS.HW.Put (Control_Output.Control);
52             Next := Next + Period;
53             STM32.Board.Red_LED.Toggle;
54         end loop;
55     end ADCS_Task;
56
57 end ADCS;
```

# ACS Hardware interface

---



The screenshot shows a software development environment with a code editor displaying Ada code. The menu bar includes File, Edit, Navigate, Find, Code, VCS, Build, SPARK, Analyze, Debug, View, Window, and Help. The toolbar contains icons for file operations like Open, Save, and Build. The code editor has tabs for various files: ttc.adb, housekeeping.adb, control.c, adcs.adb, adcs-hw.ads, adcs-parameters.ads, and adcs-hw.adb. The current tab is adcs-hw.adb. A search bar at the top right is set to 'Default'. The code itself is Ada code for the ADCS.HW package body:

```
File Edit Navigate Find Code VCS Build SPARK Analyze Debug View Window Help
ttc.adb housekeeping.adb control.c adcs.adb adcs-hw.ads adcs-parameters.ads adcs-hw.adb
git Default search

29 package body ADCS.HW is
30
31     package MF is new Message_Four_Bytes (Four_Bytes => Interfaces.IEEE_Float_32);
32
33     procedure Initialize is
34     begin
35         null;
36     end Initialize;
37
38     function Get return MGM_Readings is
39         Ingoing : aliased Message (Physical_Size => 4);
40         Data : MGM_Readings;
41     begin
42         for i in MGM_Readings'Range loop
43             Get_Fixed (COM, Ingoing'Unchecked_Access);
44             MF.Unset (Ingoing, Data (i));
45         end loop;
46         return Data;
47     end Get;
48
49     procedure Put (Output : ADCS_Vector) is
50         Outgoing : aliased Message (Physical_Size => 4);
51     begin
52         for i in ADCS_Vector'Range loop
53             -- N := (i-1)*4 +1;
54             MF.Set (Outgoing, Output (i));
55             Put (COM, Outgoing'Unchecked_Access);
56         end loop;
57     end Put;
58
59     begin
60         Initialize (COM);
61         Configure (COM, Baud_Rate => 115_200);
62     end ADCS.HW;
```

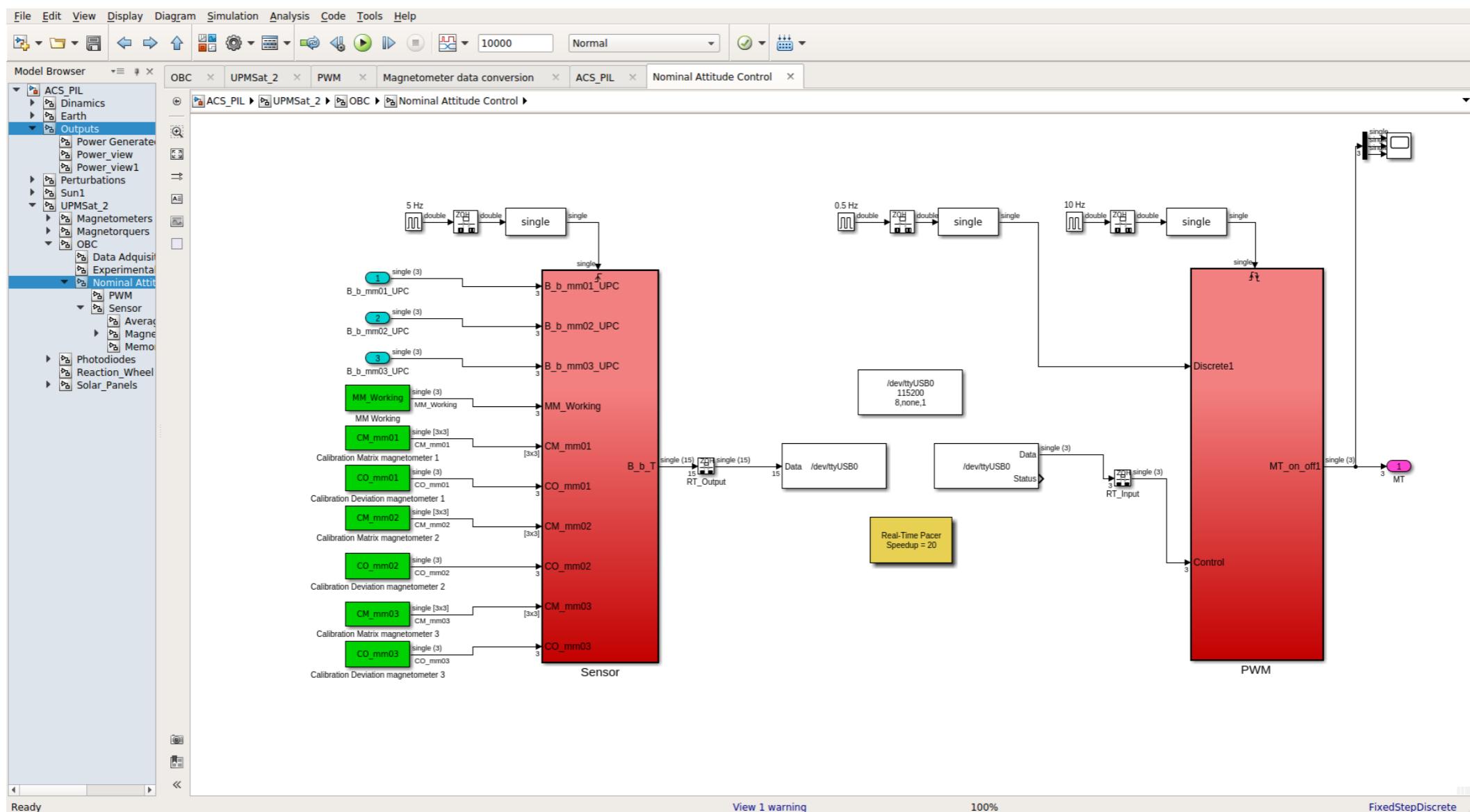
# Study the code

---

- Connect the serial line to the host PC
  - as in LAB5 but do not run PutTTY
  - serial line will be used for interchanging sensor readings and actuations
- Build the executable and debug on the board
  - as in LAB4
- Nothing will happen because OBDH is waiting host PC inputs

# Simulink model por PIL (Processor In the Loop) validation

- Open ACS\_PIL.slx from LAB7/ACS\_PIL directory
- Navigate to nominal attitude control.



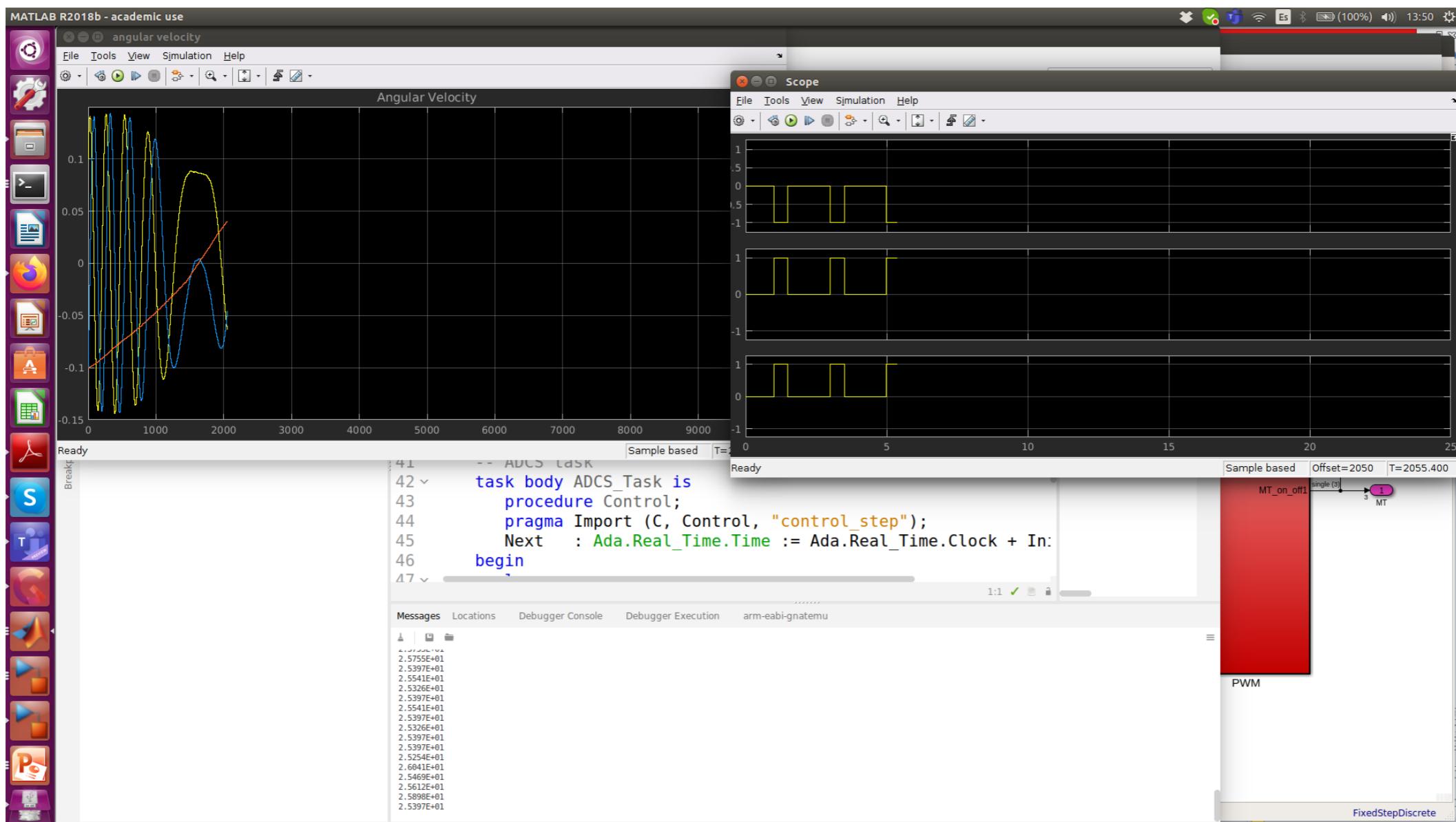
# Study the model

---

- The control block has been replaced by a connection to the serial line
  - edit the serial configuration block and put the name of the serial line of your PC
  - additional rate transition blocks have been added to ensure a proper communication
- A Real-Time pacer has been added to the model to set the simulation speed
  - Speedup = 1 should be used but it is too slow
- Start the simulation and verify angular velocity stabilization
  - Now OBDH runs and LEDs are toggled

# Simulink PIL validation

- Compare angular velocity stabilization with the original full-Simulink model.



# Make changes and play

---

- Default parameters for control blocks can be changed in Ada source file **ACS-parameters.ads**
  - `Default_omega` is the consigned angular velocity
  - `Default_MT_Working` contains the operational magneto-torques
  - In UPMSat-2 the parameters can be changed by TC
- The initial angular velocity can be changed in Simulink source code **initialization.m**
  - `omega_BI_B0 = [0.1;-0.1;-0.1];`

# Hardware In the Loop

---

- To reach a higher TRL, a more realistic operational environment is needed
  - In UPMSat-2, ADC hardware was also included in the loop
  - A SVF host with analog input/outputs had to be set up in order to generate magnetometer outputs and read magnetorquer actuations.
- Much more sophisticated operational environments are possible.
  - However, they have a high cost