



POLITÉCNICA

# Sistemas Empotrados y Ubicuos

Máster Universitario en Ingeniería Informática

## Cross-Development Environment Work

Version 1.1 — March 6, 2022

1ST STUDENT ASSIGNMENT

---

UNIVERSIDAD POLITÉCNICA DE MADRID  
DATSI

DEPARTAMENTO DE ARQUITECTURA Y TECNOLOGÍA DE SISTEMAS  
INFORMÁTICOS

<https://www.datsi.fi.upm.es>

---

© 2022 DATSI/UPM.



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. <https://creativecommons.org/licenses/by-nc-sa/4.0/>

# Contents

<b>Introduction</b>	<b>1</b>
References . . . . .	1
Acronyms . . . . .	2
<b>Overview</b>	<b>3</b>
Laboratory kit components . . . . .	3
Architecture of the laboratory platform . . . . .	3
Computer board and connections . . . . .	4
<b>Set up</b>	<b>7</b>
Install a native programming environment . . . . .	7
Install the cross-compilation tools . . . . .	8
Install MATLAB <sup>TM</sup> and Simulink <sup>TM</sup> . . . . .	11
<b>1 OBDH</b>	<b>13</b>
1.1 Software architecture . . . . .	13
1.2 Serial line connections. . . . .	14
1.3 Host terminal application. . . . .	14
1.4 Download the code and study the implementation. . . . .	16
1.5 Compile and run. . . . .	17
1.6 Make changes to the program. . . . .	17
1.7 Perform a temporal analysis of the system. . . . .	17
<b>2 OBDH with ACS</b>	<b>19</b>
2.1 Model In the Loop (MIL) validation . . . . .	19
2.2 Code generation. . . . .	21
2.3 Software architecture . . . . .	21
2.4 Compile and run with the debugger. . . . .	23
2.5 Processor In the Loop (PIL) validation. . . . .	24
2.6 Make changes to the simulation. . . . .	24
<b>3 Student assignment</b>	<b>27</b>
<b>A Temperature sensor</b>	<b>29</b>



# Introduction

This document provides instructions for laboratory work in the field of embedded systems. The laboratory is part of the Embedded and Ubiquitous Systems course of the UPM Máster Universitario en Ingeniería Informática (MUII) program. The laboratory is based on a computer kit that is used to build a simplified version of a satellite on-board software system (OBSW). An instance of the laboratory kit will be made available to every student registered in the course during the laboratory session.

Students are required to use their own personal computer, running Windows, MacOS, or GNU/Linux, to carry out the laboratory assignments. The outline of the laboratory assignments to be carried out is as follows:

1. Installation of a native programming environment.
2. Installation of the cross-platform programming tools.
3. On-board data handling (OBDH) system.
4. On-board data handling (OBDH) system with Attitude Control System (ACS).

## References

The following documents contain additional information about the software and hardware tools used to develop the work:

## Hardware

1. STMicroelectronics. DB1421 Data Brief. STM32F4DISCOVERY - Discovery kit with STM32F407VG MCU.
2. STMicroelectronics. UM1472 User manual - Discovery kit with STM32F407VG MCU.
3. STMicroelectronics. DS 8626. Data sheet - STM32F405xx, STM32F407xx. ARM Cortex-M4 32b MCU+FPU, 210DMIPS, up to 1MB Flash/192+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces & camera.
4. STMicroelectronics. RM0090 Reference manual - STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced Arm®-based 32-bit MCUs.

## Software

The following manuals are available from the “Help” menu in the GNAT Programming Studio (GPS):

1. Ada Reference Manual.
2. GPS User’s Guide.
3. GNAT User’s Guide for Native Platforms.
4. GNAT User’s Guide Supplement for Cross Platforms
5. GNAT Reference Manual.

## Acronyms

<b>ADC</b>	Analog to Digital Converter.
<b>ACS</b>	Attitude Control System.
<b>DAC</b>	Digital to Analog Converter.
<b>FPU</b>	Floating Point Unit.
<b>GCC</b>	GNU compilation system..
<b>GDB</b>	GNU Debugger.
<b>GNAT</b>	GNU Ada Translator.
<b>GNU</b>	GNU is not Unix.
<b>GPL</b>	GNU Public License.
<b>GPS</b>	GNAT Programming Studio.
<b>LGPL</b>	Lesser GNU Public License (formerly Library GPL).
<b>MCU</b>	Microcontroller Unit.
<b>OBC</b>	On-Board Computer.
<b>OBDH</b>	On-Board Data Handling.
<b>OBSW</b>	On-Board Software.
<b>OS</b>	Operating System.
<b>PC</b>	IBM Personal Computer architecture.
<b>TC</b>	Telecommand.
<b>TM</b>	Telemetry.
<b>USART</b>	Universal Synchronous Asynchronous Receiver Transmitter.
<b>USB</b>	Universal Serial Bus.

# Overview

## Laboratory kit components

The laboratory kit includes:

- An STM32F407 computer board, which emulates an on-board computer system (OBC).
- A USB A / mini USB cable which is used to connect the OBC board to the development station hosted on the student PC.
- A USB / UART interface cable which is used to provide a serial line link between the OBC board and the ground station software running on the student PC.

Figure 1 shows the components of the laboratory kit and the connections to the student PC.

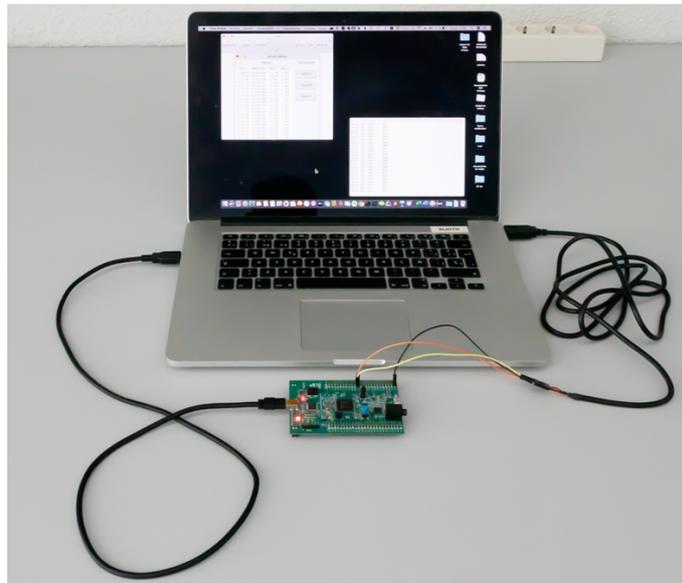


Figure 1: Laboratory kit.

## Architecture of the laboratory platform

The components of the laboratory kit are used to emulate a simplified version of a satellite on-board software system. Figure 2 shows the architecture of the laboratory system.

The system consists of a flight segment, implemented on the laboratory computer board, and a ground system, implemented on the student PC. The communication between both segments is carried out by means of a serial line, simulating the radio link of a real satellite mission. The student work is centered on programming the computer board. The ground station software will be provided by the teachers.

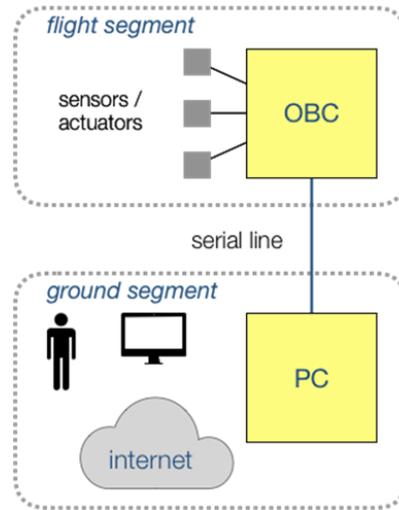


Figure 2: Architecture of the laboratory system.

## Computer board and connections

The STM32F407 board is used as a low-cost replacement for a satellite on-board computer (OBC). The board features a 32-bit ARM Cortex-M4 microcomputer, 192 KB RAM, 1 MB Flash memory and a number of other devices.

Figure 3 shows an overall view of the computer board.

The main elements that will be used in the laboratory are:

- USB ST-LINK connector, which is linked to a PC with a mini USB-USB A cable. This connection is used for the following functions:
  - power supply to the board (5 V)
  - program loading and debugging from host
- GPIO pins PB6, PB7 and GND. GPIO (General Purpose Input-Output) is a standard interface for connecting external devices. These GPIO pins are used in the laboratory to connect a serial line to a USB port on a PC, emulating the connection to the on-board radio equipment in a satellite.
- Temperature and voltage sensors. These sensors are part of the STM32 microcomputer chip, and can be read using internal registers in the MCU. They are used in the laboratory to emulate the housekeeping

USB ST-LINK mini-B connector



PB6 & PB7 pins

GND pins

USB OTG micro-AB connector

Figure 3: Computer board.



# Set up

## Install a native programming environment

The aim is to install a native programming environment for the Ada language on the student PC. This environment will later be extended with cross-compilation tools for the STM32 board to be used in the laboratory.

The programming environment to be used is GNAT Community, an open-source software development environment freely available from AdaCore, a company specialised in providing tools and solutions for developing high-integrity software,

## Download and install GNAT

The GNAT Community compilation system can be downloaded from <https://www.adacore.com/download/more>. Installation packages for Windows, MacOS and GNU Linux are available at the download page. The file README.txt provides installation instructions, which are summarised as follows:

### Windows

1. Download the file `gnat-2021-20210519-x86_64-windows64-bin.exe`
2. Run the file and follow the instructions.

### MacOS

1. Download the file `gnat-2020-20200818-x86_64-darwin-bin.dmg`
2. Open the dmg disk and execute the application inside it. In order to circumvent the system protection, control-click on the file and then click on “opens” in the emergent window.

Notice that you need to have installed the Xcode application to install GNAT. If you still see the following error:

```
ld: library not found for -lSystem
```

then you might have to execute the following:

```
xcode-select -s /Applications/Xcode.app/Contents/Developer
```

### GNU Linux

1. Download the file `gnat-2021-20210519-x86_64-linux-bin`
2. You will need to make the package executable before running it. In a command prompt, execute the following command:  
`chmod +x path_to_the_package.bin`  
and execute the package. The README.txt file contains additional installation and execution instructions.

## Test the installation with a simple program

The GNAT compilation system includes the GPS (GNAT programming studio) programming environment, which allows users to edit, compile, and run Ada and C programs. Figure4 shows the main GPS window, which is composed of the following areas:

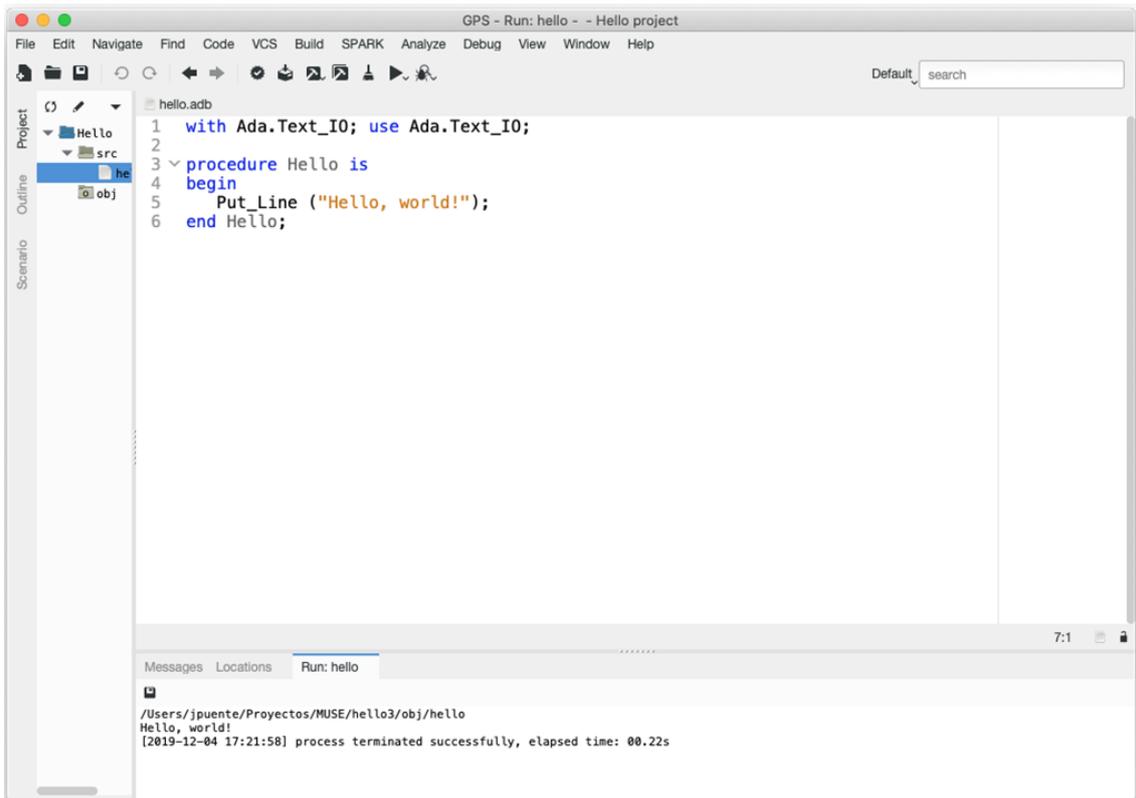


Figure 4: GNAT Programming Studio (GPS).

- a menu bar at the top
- a tool bar under the menu bar
- on the left, a notebook allowing you to switch between Project, Outline and Scenario views
- the working area in the center
- the messages window at the bottom

GPS organises source code in projects. A project is a set of source files which are compiled together in order to produce a single binary executable. Before starting you will need to create a folder to store your software projects. The recommendation is to create a folder named `OBDH_LABS` in a directory of your choice. The next activity is to write and run a simple Ada program using GPS:

1. Create a new project by clicking on `File` → `New Project ...` in the top menu. Choose the Simple Ada Project template.
2. Choose a folder to deploy the project, e.g. `OBDH_LABS/LAB1`. Set the project name to `Hello` and the main name also to `Hello`.
3. Double click on the `hello.adb` file in the project view to open the file in the working area.
4. Edit the file in the working area so that it has the same content as in figure4.
5. Build and run the executable by clicking on the `▷` symbol in the tool bar. You should see a number of compilation-related messages and, if everything is right, you will see the text “Hello, world!” in the Run tab of the bottom window.

## Install the cross-compilation tools

The aim is to get acquainted with the embedded computer board and to install and test the cross-compilation tools for GNAT that will be used to develop executable code for it.

## Cross-compilation tools

The computer board will be programmed in Ada. The GNAT cross-platform software development system will be used (figure 5), where the student PC is the host platform and the STM32 board is the target platform.

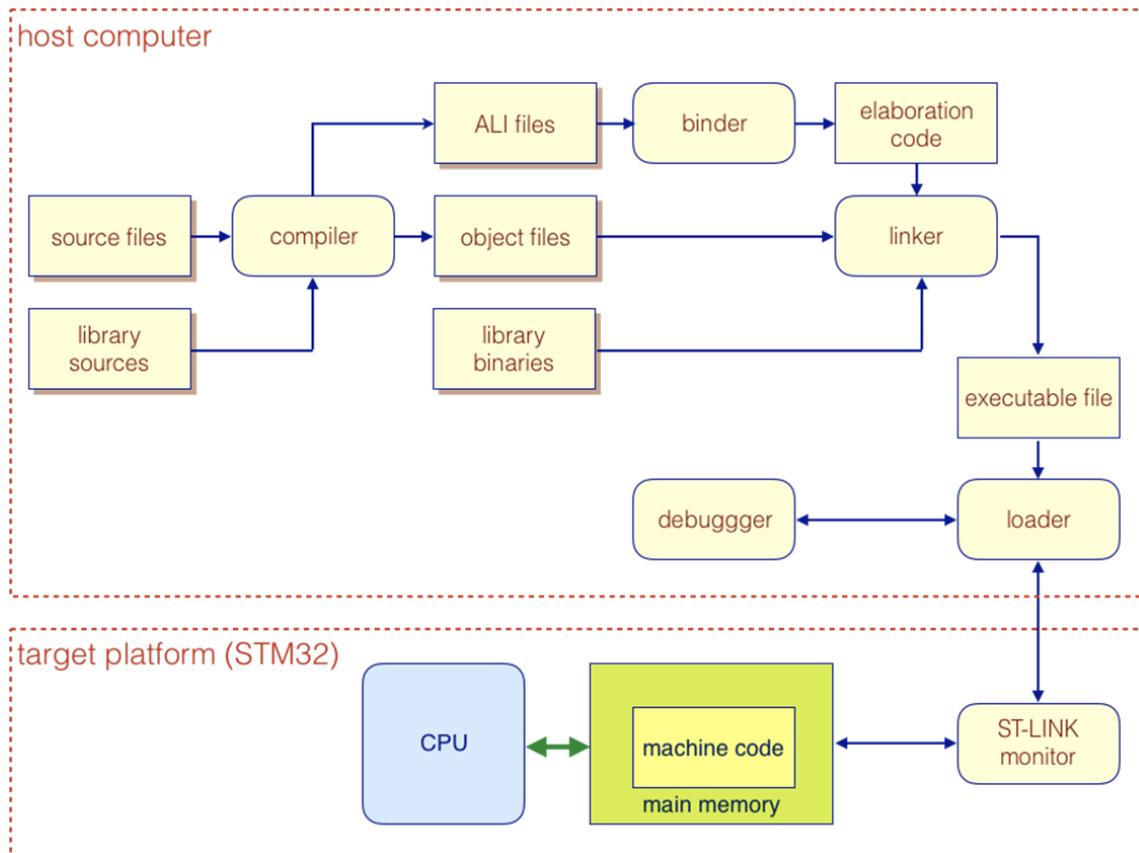


Figure 5: Cross-compilation and debugging system

In order to compile a program, the compilation chain is run on the host computer to produce an executable file for the target computer. The executable is then loaded into the target memory, from where it can be run. A monitor program is preinstalled on the target board that supports loading and debugging from the host platform.

## Download and install GNAT ARM ELF

GNAT ARM ELF is the cross-compilation chain to be used with the STM32F4 board. It can be downloaded from the same page as the native GNAT system, <https://www.adacore.com/download/more>, and there are installation packages for Windows, MacOS and GNU Linux available. The file README.txt provides installation instructions, which are summarised as follows.

### Windows

1. Select ARM ELF (hosted on windows64) and download the file gnat-2021-20210519-arm-elf-windows64-bin.exe
2. Run the file and follow the instructions.
3. You will also need to install the USB driver for the ST-LINK probe. To do so, go to [http://www.st.com/content/st\\_com/en/products/embedded-software/development-tool-software/stsw-link009.html](http://www.st.com/content/st_com/en/products/embedded-software/development-tool-software/stsw-link009.html), and click on Get Software. Click on Get Software under the Download column of the table that shows up to obtain the driver. You will need to accept ST Micro's license agreement and enter your contact details. Once downloaded unzip the USB device driver and run the installer, accepting all the defaults.

## MacOS

1. Download the file `gnat-community-2019-20190517-arm-elf-darwin-bin.dmg`
2. Open the dmg disk and execute the application inside it. In order to circumvent the system protection, control-click on the file and then click on “open” in the emergent window.
3. You will also need the `st-util`, `st-flash`, and `st-info` tools. You can download the binaries from <https://github.com/texane/stlink/releases/download/1.3.0/stlink-1.3.0-macosx-amd64.zip>. Unzip and copy the files in the `bin` directory to a directory in your `PATH`. You may need to circumvent MacOS protection by executing the command:
 

```
$ xattr -d com.apple.quarantine path-to-executable-file
```

## GNU Linux

1. Download the file `gnat-2021-20210519-arm-elf-linux64-bin`
2. You will need to make the package executable before running it. In a command prompt, execute the following command:
 

```
chmod +x path_to_the_package.bin
```

 and then execute the package.
3. You will also need to install the `stlink` tools. In Ubuntu and Debian `stlink` must be installed from sources. Follow the instructions on [http://docs.adacore.com/live/wave/gnat\\_ugx/html/gnat\\_ugx/gnat\\_ugx/arm-elf\\_topics\\_and\\_tutorial.html#linux](http://docs.adacore.com/live/wave/gnat_ugx/html/gnat_ugx/gnat_ugx/arm-elf_topics_and_tutorial.html#linux).

The `README.txt` file contains additional installation and execution instructions.

## Test your installation with an embedded program

The next thing is to compile and run a simple embedded program. This program is only intended to test that the compilation chain and the ST-LINK tools have been properly installed.

Open GPS and do the following:

1. Create a new project by clicking on `File` → `New Project ...` in the top menu. Choose the `STM324F compatible` → `LED demo project template`.
2. Choose a folder to deploy the project, e.g. `OBDH_LABS/LAB2`. Set the project name to `led_demo` and the main name to `main`. A window with a project including a number of source files will open.
3. Right-click on the project icon on the left side area, and choose `Project` → `Properties` (figure 6). On the emerging window, select `Embedded` and change the `Connection tool selector` to `st-util`. Save the settings.
4. Connect the `STM32F4` board to the computer by means of a `USB A / mini USB` cable.
5. Build the executable and load it into the board by clicking on the  symbol in the tool bar (or select `Build` → `Bareboard` → `Flash to board` on the top menu). You should see a number of compilation-related messages ending with “Flashing complete. You may need to reset or cycle power”.
6. If everything is all right, you will see the LEDs on the board blinking in a circular pattern.
7. Download and install the Ada Drivers Library The Ada Drivers Library is a set of Ada packages that make it easier to write software for embedded devices, including the `STM32F4` microcontroller family and some demonstration boards. The source code can be found at [https://github.com/AdaCore/Ada\\_Drivers\\_Library](https://github.com/AdaCore/Ada_Drivers_Library). To install the library, click on the green `Clone or download` button on the upper right side and then on `Download Zip` in the emerging window. You will get a zip archive in your downloads folder. Unzip the archive and move the resulting folder to your `OBDH_LABS` folder. Rename the folder to `Ada_Drivers_Library`, removing any trailing text.
8. Compile and run a test program with the Ada Drivers Library

Open GPS and do the following:

1. Select Open project on the welcome window. Navigate to .../OBDELABS/Ada\_Drivers\_Library/examples/STM and open the project file: blinky\_f4disco.gpr.
2. Build the executable and load it into the board by clicking on the  symbol in the tool bar (or select Build → Bareboard → Flash to board on the top menu). When the loading is complete, you will see the board LEDS blinking all at the same time.

## Install MATLAB<sup>TM</sup> and Simulink<sup>TM</sup>

MATLAB and Simulink will be used to generate C code from a Simulink model and to validate the system by Processor In the Loop (PIL).

UPM has a campus license available for students. [http://www.upm.es/sfs/Rectorado/Vicerrectorado%20de%20Tecnologias%20de%20la%20Informacion%20y%20Servicios%20en%20Red/Servicio%20de%20Planificacion%20Informativa%20y%20Comunicaciones/SW/MATLAB\\_UPM\\_Estudiantes.pdf](http://www.upm.es/sfs/Rectorado/Vicerrectorado%20de%20Tecnologias%20de%20la%20Informacion%20y%20Servicios%20en%20Red/Servicio%20de%20Planificacion%20Informativa%20y%20Comunicaciones/SW/MATLAB_UPM_Estudiantes.pdf) explains how to install MATLAB and Simulink.

In order to generate C code, the Embedded Coder toolbox and its dependencies must be installed.



# Assignment 1

## OBDH

The aim of this assignment is to experiment with a reduced version of an On-Board Software (OBSW) of a satellite. The OBSW reads the MCU temperature by using a temperature sensor that is connected to an ADC.

### 1.1 Software architecture

The software architecture of the OBSW is depicted in 1.1 The software components are:

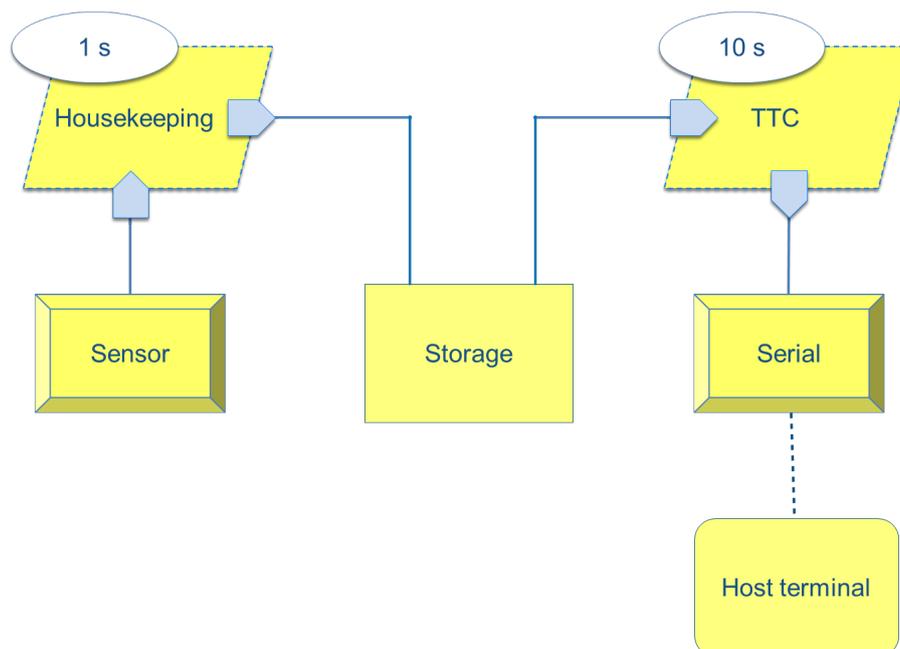


Figure 1.1: Software architecture of OBSW.

**Telemetry and Telecommand (TTC).** This module is in charge of the communications with the ground station. The ground station is simulated by the Host terminal. It must be activated cyclically with a 10 second period.

**Serial.** This module provides a high-level interface to a text console where the measured temperature values can be output.

**Housekeeping.** Main component, which performs the basic functionality of the system, i.e. reading a temperature value and storing the value on the Storage component. It must be activated cyclically with a 1 second period.

**Sensor.** This module provides a high-level interface to the temperature sensor and deals with all the details of reading the ADC to which the sensor is connected.

**Storage.** This component is a data object storing one temperature value, which is written by Housekeeping and read by TTC.

Since the OBC board does not have a text output device, temperature values are sent by a serial line to the ground station. In this way, the radio link between the satellite and the ground station is simulated.

## 1.2 Serial line connections.

This scheme makes use of the USB/UART interface cable provided to the students. The USB/UART cable has a TTL connector that must be connected to the STM32f4 board pins that convey the serial line (UART) signals (figure 1.2).



Figure 1.2: UART cable connector.

The connections to be made are summarized in the following table (see figure 3 for the location of the pins on the board):

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

Table 1.1: Serial line connections on board.

The other end of the interface cable has a USB-A connector that must be plugged to a USB port on the host computer. The values sent to the host computer are displayed using a terminal application that can handle a USB serial port. The host terminal application should be set to taking the USB serial port as input with a transmission rate of 115200 bps and 8N1.

## 1.3 Host terminal application.

### 1.3.1 Windows

The recommended application to display messages received on the USB serial port is PuTTY. You can download an installation package from <https://www.putty.org>.

In order to configure the application, you need first to identify the COM port corresponding to the USB serial line. Open the Device Manager and look at the USB Serial Port entry. The COM port is displayed next to it (e.g. COM 4 in figure 1.3).

Now, to set up PuTTY, open the application and set the configuration parameters as shown in figure 1.2.

### 1.3.2 MacOS

The recommended application is screen, which is already installed in MacOS. First you have to identify the USB serial port. Open a terminal window and type

```
$ ls /dev — grep -i usb
```

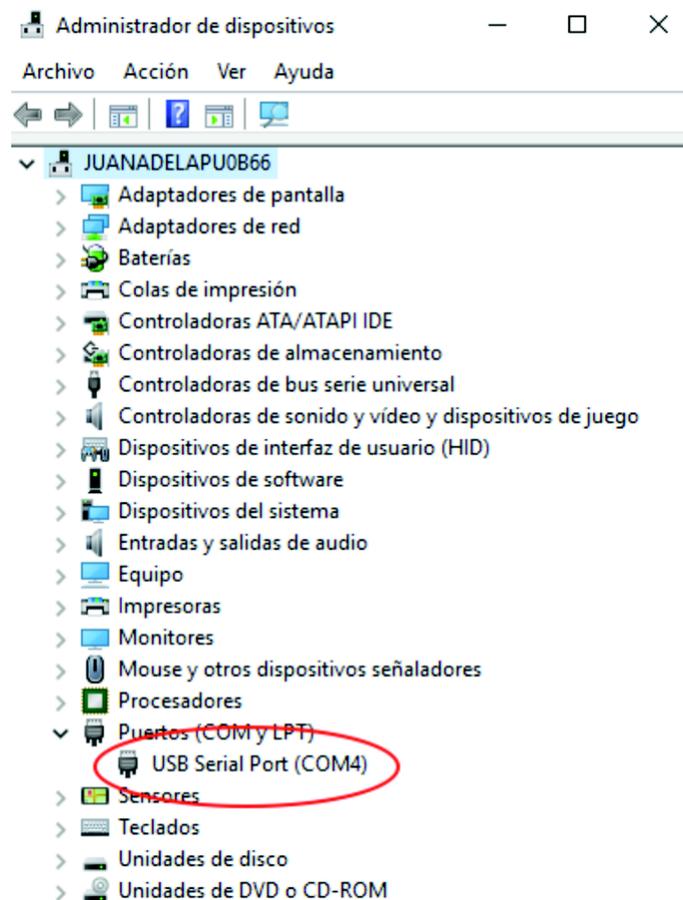


Figure 1.3: Identification of usb serial port.

You will get a list of devices like the following:

```
cu.usbserial-FTA5I24G
tty.usbserial-FTA5I24G
```

As you can see, there are two devices for each serial line. You can use any of them, but for reasons not to be discussed here it is better, in general, to use the one starting with cu.

To use the screen application enter the following command:

```
$ screen /dev/cu.usbserial-XXXX 115200
```

where /dev/cu.usbserial-XXXX is the name of your device.

To exit the application, type CTRL-A and then CTRL-K.

### 1.3.3 GNU Linux

The recommended application is screen,<sup>4</sup> which can be installed in Ubuntu Linux with:

```
$ sudo apt install screen
```

In order to identify the USB serial port, type the following command on a terminal:

```
$ ls /dev — grep -i usb
```

You will get a result like the following:

```
ttyUSB0
```

To use the screen application enter the following command:

```
$ screen /dev/ttyUSB0 115200
```

To exit the application, type CTRL-A and then SHIFT-K.

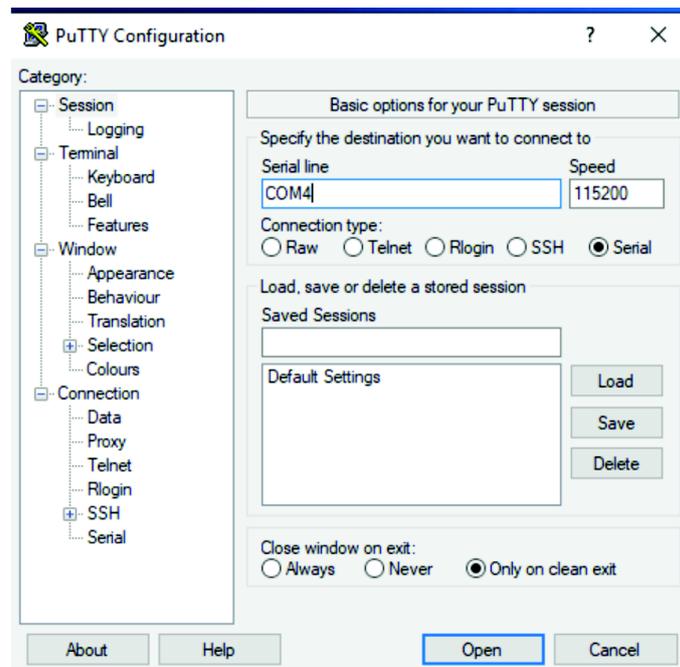


Figure 1.4: PuTTY configuration.

## 1.4 Download the code and study the implementation.

The implementation code, as initially provided to the students, can be downloaded from [https://github.com/STR-UPM/OBDH\\_LABS](https://github.com/STR-UPM/OBDH_LABS). Click on Clone or download, download a zip archive, unzip and move to your work directory. The code for this assignment is in the LAB6 folder.

The Housekeeping package is the root element of the housekeeping subsystem. Its specification consists of one procedure, Initialize, that starts the operation of the component. It has four subpackages:

**Housekeeping** is the root package of the subsystem and contains a concurrent task, Housekeeping\_Task that stores Data in Storage and toggles the blue LED every second.

**Housekeeping.Data** contains the definitions of the data types used in the subsystem. The data type Analog\_Data is used to read the ADC, which are integers in the range 0 to 4095 as directly provided by the ADC. The data type State is a record that contains the ADC read and the corresponding timestamp.

These ADC values have to be converted to engineering units. i.e. degrees Celsius, by following the specification of the temperature sensor (see appendix ??). Raw sensor readings are usually sent to ground station that is in charge of converting them to engineering units.

**Housekeeping.Sensor** contains the details of the temperature sensor. Its specification includes the Initialize and Get procedures. This package uses the Ada Drivers Library to interact with the OBC board hardware.

**Housekeeping.Images** includes functions which are used to convert to Strings temperature and timestamp values.

The TTC package is the root of the telecommunications system, which in this version is greatly simplified with respect to a real application. It contains a concurrent task, HK\_Task, which takes measured sensor values from Storage and sends them to the ground station by using Serial every ten seconds. The task also toggles the orange LED.

The Storage package implements the communication between the Housekeeping and TTC subsystems. Since this object is shared by two concurrent tasks, it is implemented as a protected object, so that its operations are executed in mutual exclusion. There is also conditional synchronization: the TTC task must wait until there is a fresh value in the store. However, Housekeeping should not wait if the previous value put into Storage has not been consumed, in order not to delay

the housekeeping function. In this case, the stored value is overwritten. Notice that this differs from the classical specification of a bounded buffer.

The Serial component is implemented by the Serial.IO package and other packages in the serial\_ports folder. These packages have been adapted from the examples in the Ada Drivers Library. The blocking kind of serial port has been chosen for this project. This means that the task calling the Put operation (TM\_Task) waits on a busy loop until the operation is complete.

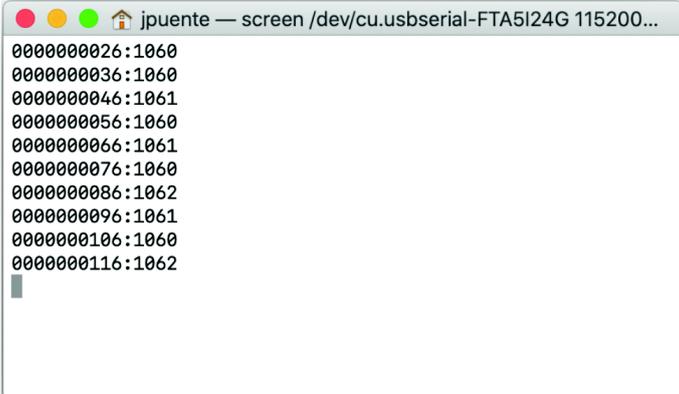
The main procedure is OBSW. It calls Housekeeping.Initialize, which initializes the sensor so that Housekeeping.Task can proceed. Additionally, the green LEDs is toggled on and off to provide a visual check that the program is running.

Notice that Run, and hence Initialize and OBSW, never return. Therefore the program executes indefinitely, as is common in embedded systems.

## 1.5 Compile and run.

Open GPS and do the following:

1. Select Open project on the welcome window. Navigate to the OBSW directory and open the realtime.housekeeping.gpr project file.
2. Build the executable and load it into the board by clicking on the  symbol in the tool bar (or select Build → Bareboard → Flash to board on the top menu).  
The program will be compiled, and the executable will be loaded into the board flash memory. After that, the program starts to run on the board (check the blinking LEDs).
3. Connect the serial cable to a USB port on the host computer, if not already done.
4. Identify the serial port name on the host computer and launch the remote terminal application as explained in section 1.3. The sensor measured values together with their respective timestamps will start being displayed on the host application (figure 1.5).



```

jpuente — screen /dev/cu.usbserial-FTA5I24G 115200...
000000026:1060
000000036:1060
000000046:1061
000000056:1060
000000066:1061
000000076:1060
000000086:1062
000000096:1061
000000106:1060
000000116:1062

```

Figure 1.5: Sample output on host terminal.

## 1.6 Make changes to the program.

You may make change to the provided program in order to make sure that you understand the logics behind the source code. Proposed change is:

Include the conversion to Celsius in the TTC.Send procedure. Temperature function of HK\_data-converter that can be found in utilities can be used.

## 1.7 Perform a temporal analysis of the system.

In order to carry out a response-time analysis of the temporal behaviour of the system, you will need to measure the execution time of the task bodies and the protected procedure bodies. A simple loop technique using the standard real-time clock will be enough for this assignment.

An execution time measurement tool is available in the LAB6 directory. In order to use it, perform the following steps:

1. Open GPS and select Open project on the welcome window. Navigate to the LAB6 directory and open the `wcet_meter.gpr` project file.
2. Build the executable and load into the board in the same way as for the `realtime_housekeeping.gpr` project.
3. Make sure that the serial cable is still connected to the board and the USB port in the host computer. If the remote terminal application is not open, open it.

A measurement test is executed on the board, and repeated every 60 s. The output of the test is shown on the host terminal application (figure 1.6). The output shows the execution times for the bodies of the Housekeeping (HK) and TTC (TC) tasks, as well as the bodies of the protected operations of the Storage object (ST). Notice that a new entry, `Get_Immediate`, has been added for the latter in order to avoid the measuring task to get blocked. The new entry is exactly the same as `Get` but has a `True` barrier so that it is always open.

```

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

```

Figure 1.6: Output of `wcet` measurement tool.

In the example shown on figure 1.6, the HK execution time has been measured  $10^6$  times, with a total measurement time of 13.02 s. Therefore, the value to be taken for the response time analysis is  $13.02 \cdot 10^{-6} \text{ s} = 13.02 \mu\text{s}$ , and the same for the other tasks. Take into account that the values measured on your board will probably be slightly different from the above shown.

Once you have an estimate of worst case execution times, apply the RTA equations for computing the worst-case response time and check if all the deadlines are met. The setup for the calculations is shown on table 1.2.

Task	T	C	B	D	R	P	Storage	Operation
Housekeeping	1.0	$13 \cdot 10^{-6}$	?	1.0	?	20	$3 \cdot 10^{-6}$	CPut
TTC	10.0	$26 \cdot 10^{-6}$	?	2.0	?	10	$4 \cdot 10^{-6}$	CGet
						CP		?

Table 1.2: Data arrangement for RTA of the housekeeping system.

Temporal analysis will be explained later during the course. Therefore, execution times should be stored and the temporal analysis will be performed later.

# Assignment 2

## OBDH with ACS

The aim of this assignment is to validate the Attitude Control System (ACS) with Processor In the Loop (PIL). The reduced version of an On-Board Software (OBSW) is used with the ACS of the UPMSat-2 satellite. The ACS uses magnetic sensors and actuators, which is commonly known as magnetic attitude control (figure 2.1).

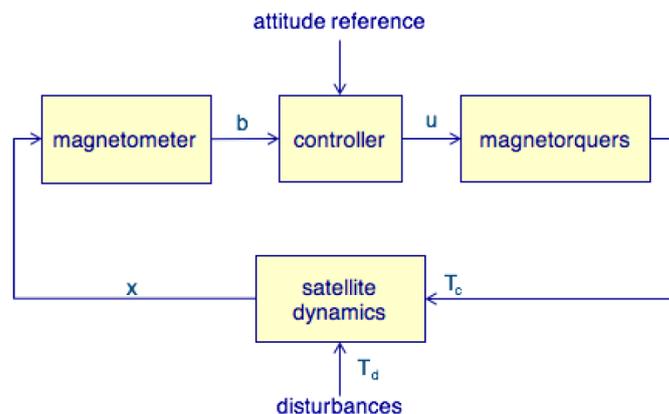


Figure 2.1: Magnetic attitude control system.

Magnetometers are magnetic sensors that provide a measurement of the strength and direction of the magnetic field, i.e. the magnetic field vector, at a given point. Magnetorquer are magnetic coil which produce a magnetic moment that interacts with the Earth's field, thus enabling the attitude of the satellite to be changed.

### 2.1 Model In the Loop (MIL) validation

Software validation usually includes testing the system under real operating conditions. However, for obvious reasons, on-board space software as well as many other embedded systems cannot be tested in this way. Simulation models are commonly used in these cases.

The first validation phase uses 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 (figure 2.2). This is usually carried out by a control engineer using a simulation tool. Simulink is commonly used for ACS development.

The ACS simulink model can be simulated by running matlab from directory LAB7/ACS and opening ACS.slx. Three new windows will be pop-up: the simulink window with the ACS model and two scope windows that show the angular velocity of the satellite in body reference and the actuation over the three magnetorquers.

The simulink window (figure 2.3) shows the high level blocks, that are the satellite itself, its dynamic and the models of the Earth's field and Sun with the perturbations.

The nominal attitude control can be show by selecting Nominal Attitude Control in the Model Browser menu (left part of simulink window) or by clicking on Satellite → OBC → Nominal Attitude Control blocks.

Nominal attitude control has three blocks (figure 2.4):

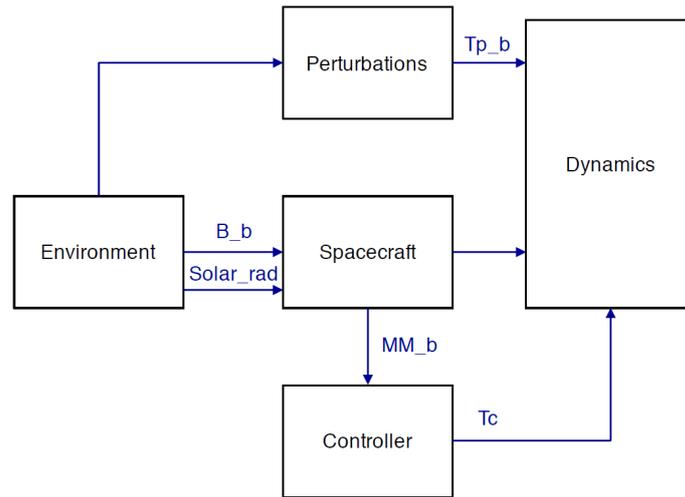


Figure 2.2: UPMSat-2 ACS high level model view.

**Sensor** samples the analog inputs of the magnetometers. The inputs are converted to engineering units using calibration data.

**Control** implements the attitude control law that computes the control action to be output to the magnetorquers.

**Actuator** activates the magnetorquers according to the computed control action.

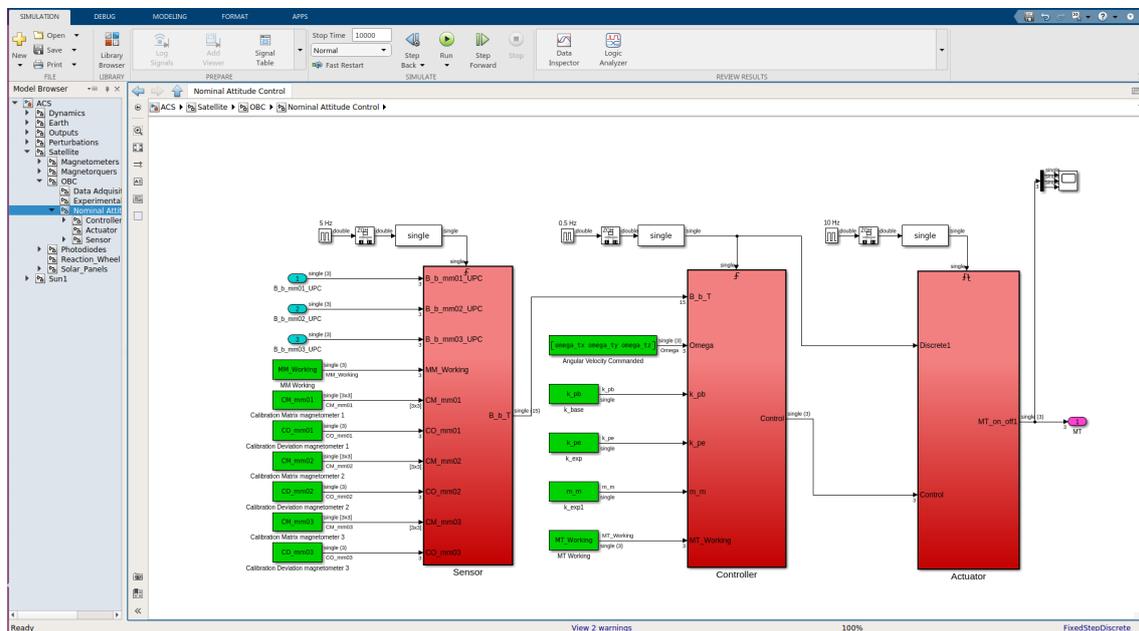


Figure 2.4: Nominal attitude control.

To simulate the model and verify its behaviour, click on Run bottom. The evolution of the angular velocity of the satellite and the actuation over the three magnetorquers will be shown in the corresponding scope windows. The commanded angular velocity is  $[0, 0, 0.1]$  rad/s and the result of the simulation (figure 2.5) shows that evolution from the initial angular velocity  $([0.1, -0.1, -0.1]$  rad/s).

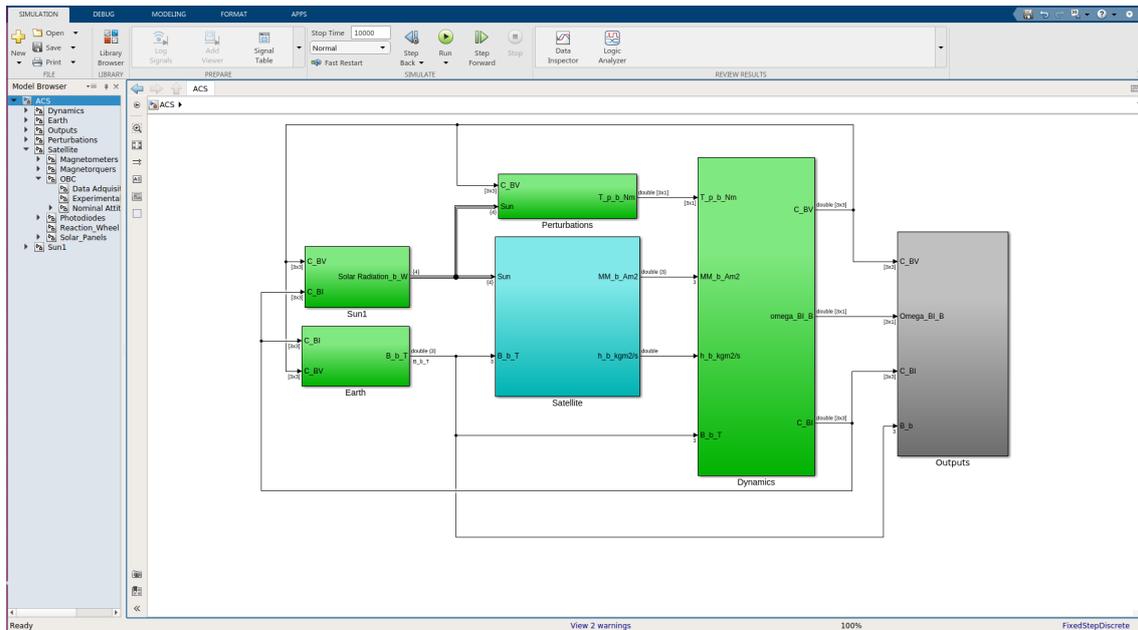


Figure 2.3: UPMSat-2 simulink model.

## 2.2 Code generation.

The next step is execute the ADCS on hardware. In this assignment, only the Control block will be execute on the target board. The corresponding code can be generated by using the Embedded Coder toolbox but it is needed to isolate Control block from the ACS model. It can be done by clicking in the Control block, selecting all the block content (except the trigger block) and saving it in a new model.

This model named control.slx can be found in LAB7/ACS\_PIL directory. Use matlab to open it and then select APPS in the top menu, Embedded Coder will appear. If not, it must be installed by clicking Get Add-Ons and searching it. The Embedded Coder window (figure 2.6) will appear after clicking on Embedded Coder icon.

The code generation option as well as characteristics of the target hardware can be set by clicking on the Settings menu. control.slx model has already the proper options, therefore you can take a look but be carefully and do not modify then.

Now the code can be generated by clicking on the Build menu. Once upon the code is generated, a code generation report window appear. It is possible to explore the generated code together with different code metrics. Click on control.h and look for lines 50-79 (figure 2.7) where the generated code interface is located.

There are two record type definitions (struct) called ExternalInputs and ExternalOutputs that are used to interchange data with the blocks Sensor and Actuator (figure 2.4). Data are interchanged with two objects: rtU and rtY. Moreover, function control\_initialize initializes the control code and function control\_step performs the control algorithm.

The generated code will be embedded in the OBSW by taking into account this interface.

## 2.3 Software architecture

The implementation code, as initially provided to the students, can be downloaded from [https://github.com/STR-UPM/OBDH\\_LABS/LAB7](https://github.com/STR-UPM/OBDH_LABS/LAB7).

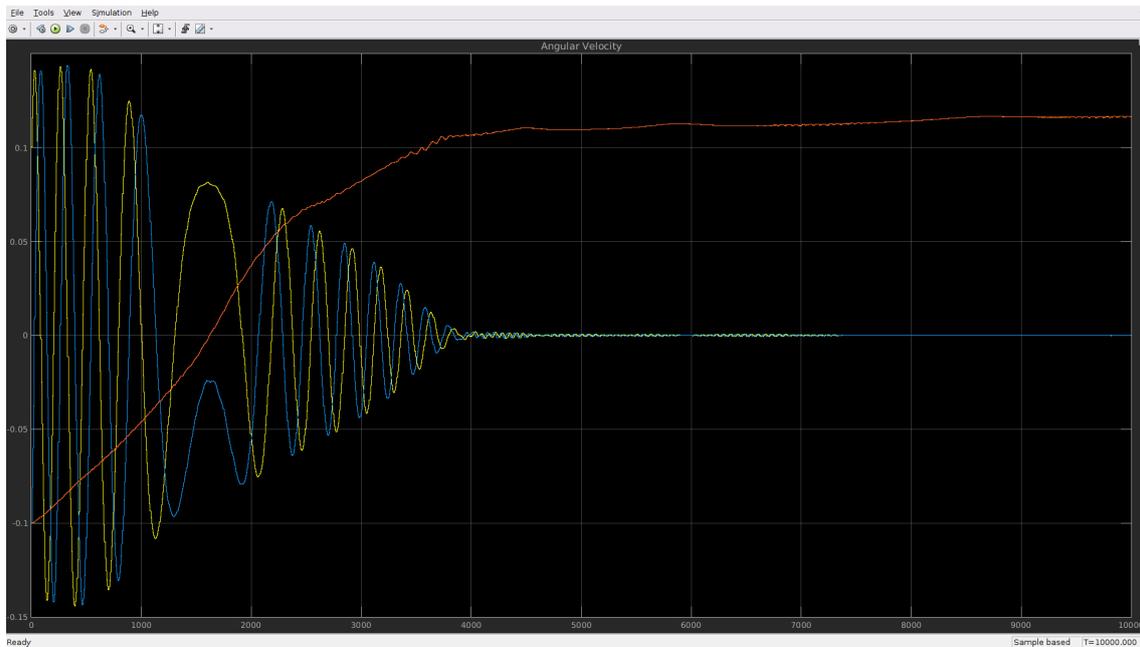


Figure 2.5: Angular velocity evolution.

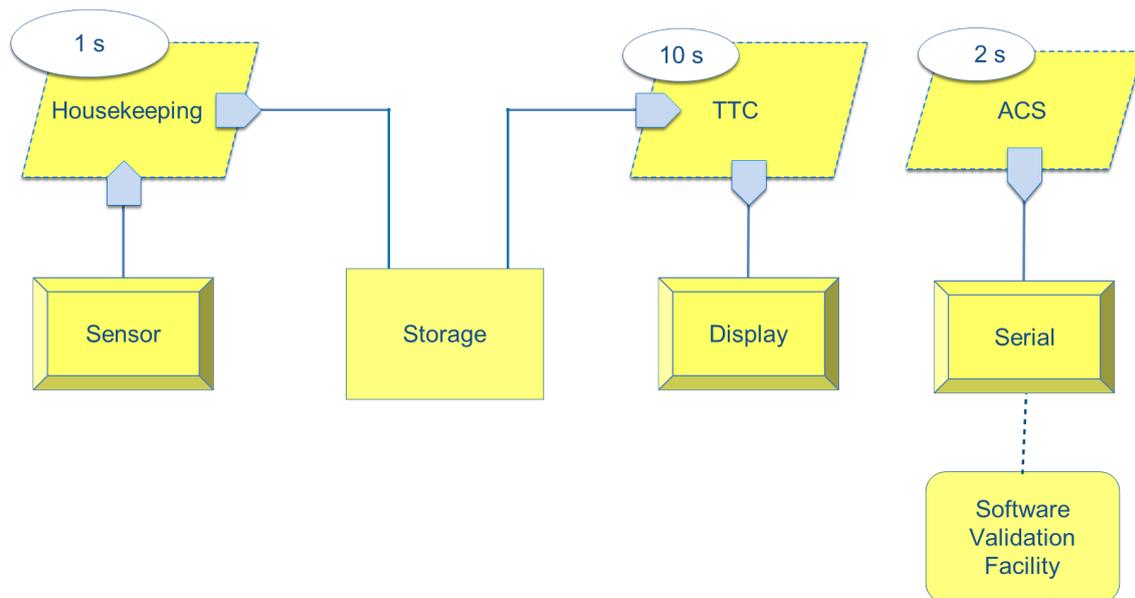


Figure 2.8: Software architecture of OBSW with ACS.

The software architecture of the OBSW with ACS is depicted in figure 2.8 and the differences with the previous architecture (figure 1.1) are:

The ADCS package is the root element of the ADCS. Its specification consists of one procedure, Initialize, that starts the operation of the component. It has three subpackages:

**ADCS** is the root package of the subsystem and contains a concurrent task, ADCS\_Task that reads the magnetic field vector, calculates the torque vector and sends it to magnetorquers. It also toggles the red LED every two second.

The body of this package uses the generated code by setting the inputs, calling the functions and retrieving the outputs following the interface of figure 2.4. ADCS\_Task performs the control algorithm by calling control\_step. This is shown in figure 2.9. It uses Export and Import pragmas to interface the generated C code.

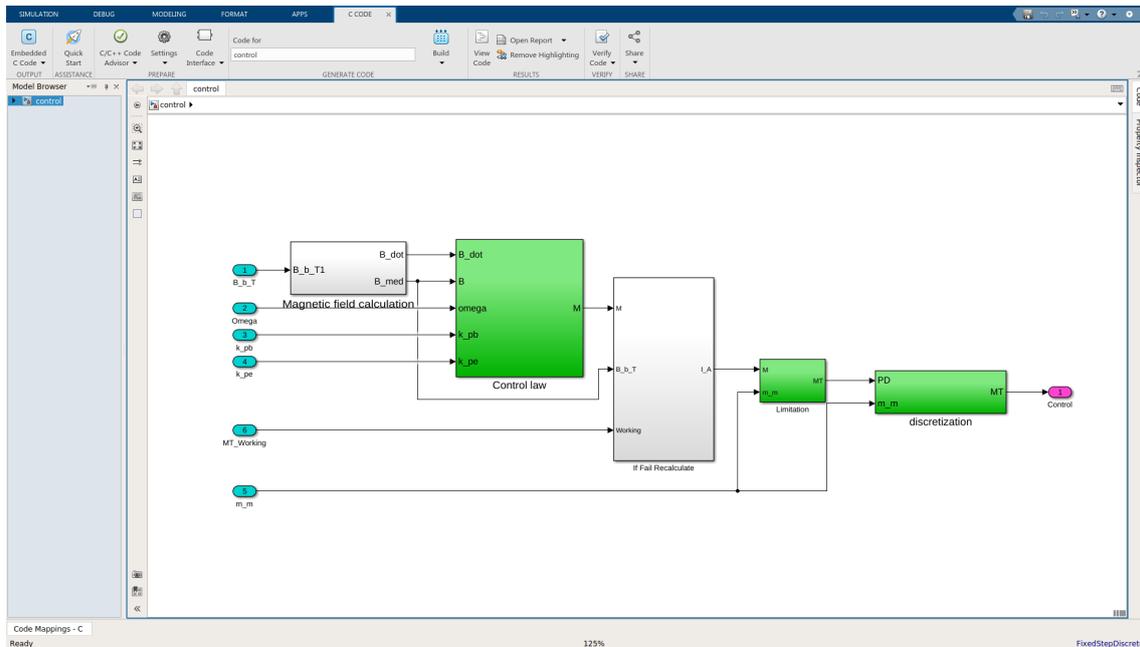


Figure 2.6: Embedded Coder toolbox.

**ADCS.Parameters** contains the definitions of the data types used in the subsystem and parameters that are used to tune the control algorithm. These parameters can be changed by telecommand in the UPMSat-2 OBSW. The data type `Controller.Input` is used to read the magnetic field vector in teslas, which are IEEE single precision float numbers. The data type `Controller.Output` are used to send the actuation to magnetorquers in newton-meters, which are IEEE single precision float numbers. These data types correspond to the generated C structs `ExternalInputs` and `ExternalOutputs` (see figure 2.7).

**ADCS.HW** is in charge of getting the magnetic field vector and putting the torques. It hides the details of the hardware. Its specification includes the `Put` and `Get` subprograms. This package uses the serial port to interchange magnetic field and torque values with the Software Validation Facility.

The Software Validation Facility (SVF) is an auxiliary computer, linked to the OBC by a serial line, to run a simulation model of the Earth's magnetic field and satellite dynamics. In this way, engineering values (Nm and T) can be interchanged.

Host computers are also used as SVF by executing a Simulink<sup>TM</sup> model of the Earth's magnetic field and satellite dynamics.

In this assignment, the serial line is used to interchange data between ADCS and SVF. Therefore, housekeeping telemetry are send to a text console that is simulated on the host computer, using a mechanism called semihosting. When the target board is connected to the host by means of the ST-LINK USB cable, and the embedded program is run using the debugger in the host, the standard output is re-directed to the debugger console. The GPS environment supports semihosting.

## 2.4 Compile and run with the debugger.

Open GPS and do the following:

1. Select Open project on the welcome window. Navigate to the OBSW directory and open the `realtime.housekeeping.gpr` project file.
2. Build the executable and load it into the board by clicking on the  symbol in the tool bar (or select Build → Bareboard → Debug on board on the top menu).

The program will be compiled, and the executable will be loaded into the board memory by the debugger. The debugger console (lowest window in GPS) shows the following lines:

The screenshot shows a code generation report window with a sidebar on the left and a main text area on the right. The sidebar contains a 'Contents' section with links to 'Summary', 'Subsystem Report', 'Code Interface Report', 'Traceability Report', 'Static Code Metrics Report', 'Code Replacements Report', and 'Coder Assumptions'. Below this is a 'Generated Code' section with expandable folders for 'Main file' (containing 'ert\_main.c'), 'Model files' (containing 'control.c' and 'control.h'), and 'Utility files (1)'. The main text area displays C code with line numbers 50 through 79. The code defines external inputs and outputs for a control system, including arrays for real32\_T signals and extern declarations for control initialization and stepping functions. Comments indicate the root paths for various signals like B\_b\_T, Omega, k\_pb, k\_pe, m\_m, and MT\_Working. The code ends with a /\*- comment on line 79.

Figure 2.7: Code generation report.

```
...
(gdb) monitor reset halt
(gdb)
```

3. Type continue or just c on the debugger console (or select Debug → Continue on the top menu).

```
(gdb) c
Continuing.
[program running]
```

After that, the program starts to run on the board and temperature reads are displayed on Messages tab of the debugger window. However, the Red LED does not blink because ACS is waiting sensor inputs from SVF.

## 2.5 Processor In the Loop (PIL) validation.

The SVF shall provide sensor inputs and retrieve magnetorquer outputs. To do that, the remain part of the original simulink model will be used, i.e. all the blocks except the Sensor block.

This model named ACS\_PIL.slx can be found in LAB7/ACS\_PIL directory. Open it and again three new windows will be pop-up: the simulink window with the ACS\_PIL model and two scope windows that show the angular velocity of the satellite in body reference and the actuation over the three magnetorquers.

The Control block of this model has been substituted by serial link connections as shown in figure 2.10. Identify the serial port name on the host computer and edit the serial configuration block by selecting the serial line of your PC.

Additional rate transition blocks has been added for a proper communication and a Real-Time Pacer block has been also added to set the simulation speed. In a real case, an speedup equal to 1 should be used but it is too slow.

Start the simulation and verify angular velocity stabilization. Now ADC runs and red LED is toggled.

## 2.6 Make changes to the simulation.

- Default parameters for control blocks can be changed in Ada source file ACS-parameters.ads.

```

23 package body ADCS is
24
25     -- controller block input and output objects must be global
26     Control_Input : Controller_Input;
27     pragma Export (C, Control_Input, "rtU");
28
29     Control_Output : Controller_Output;
30     pragma Export (C, Control_Output, "rtY");
31
32     -- Initialize the ADCS subsystem
33 procedure Initialize is
34     procedure Control_Initialize;
35     pragma Import (C, Control_Initialize, "control_initialize");
36 begin
37     Control_Initialize;
38 end Initialize;
39
40 -- ADCS task
41 task body ADCS_Task is
42     procedure Control;
43     pragma Import (C, Control, "control_step");
44     Next_Time : Ada.Real_Time.Time := Ada.Real_Time.Clock + Initial_Delay;
45 begin
46     loop
47         delay until Next_Time;
48         Control_Input.B_b_T := ADCS.HW.Get;
49         Control;
50         ADCS.HW.Put (Control_Output.Control);
51         STM32.Board.Red_LED.Toggle;
52         Next_Time := Next_Time + Period;
53     end loop;
54 end ADCS_Task;
55 end ADCS;

```

Figure 2.9: Implementation of ACS package.

- Default\_omega is the consigned angular velocity.
- Default\_MT\_Working contains the operational magneto-torques.

In UPMSat-2 many parameters can be changed by TC.

- The initial angular velocity can be changed in Simulink source file initialization.m.
  - $\text{omega\_BLB0} = [0.1;-0.1;-0.1];$



# Assignment 3

## Student assignment

Students have to write a brief report of about 2-3 pages about the work done during the session. This report must include an analysis about cross-development environment for embedded systems and the main differences with a native environment.

The schedulability analysis proposed in section 1.7 must be included. The report can contain recommendations to improve the assignment and personal opinions.

If you have made the proposed changes or any other, you should write them in the report and upload the code too.

Moodle is the selected platform for uploading assignment and the dead line is December 10th.



# Appendix A

## Temperature sensor

The STM32F407 reference manual (section 13.10) states that the internal temperature sensor of the MCU is internally cabled to the ADC1\_IN16 analog input channel. The steps required to read the sensor are:

1. Select ADC1\_IN16 input channel in the ADC.
2. Select a sampling time greater than the minimum sampling time specified in the datasheet (see table A.1 below).
3. Set the TSVREFE bit in the ADC\_CCR register to wake up the temperature sensor from power down mode.
4. Start the ADC conversion by setting the SWSTART bit (or by external trigger).
5. Read the resulting VSENSE data in the ADC data register.
6. Calculate the temperature using the following formula:

$$\text{Temperature (in } ^\circ\text{C)} = (\text{VSENSE} - \text{V25}) / \text{Avg\_Slope} + 25$$

Where:

- V25 = VSENSE value for 25 °C (table A.1)
- Avg\_Slope = average slope of the temperature vs. VSENSE curve (table A.1).

The sensor has a startup time after waking from power down mode before it can output VSENSE at the correct level. The ADC also has a startup time after power-on, so to minimize the delay, the ADON and TSVREFE bits should be set at the same time.

The sensor has a range of -40 to 125 °C, with a precision of  $\pm 1.5$  °C. Its main characteristics are described in the STM32F407 datasheet (table A.1).

Symbol	Parameter	Min	Typ	Max	Unit
TL	VSENSE linearity with temperature	-	$\pm 1$	$\pm 2$	°C
Avg.Slope	Average slope	-	2.5		mV/°C
V25	Voltage at 25 °C	-	0.76		V
tSTART	Startup time	-	6	10	$\mu\text{s}$
TS_temp	ADC sampling time when reading the temperature (1 °C accuracy)	10	-	-	$\mu\text{s}$

Table A.1: STM32F407 temperature sensor characteristic.

The Ada Drivers Library includes the package STM32.ADC, which provides facilities for handling the analog to digital converter.