MÁSTER UNIVERSITARIO EN SISTEMAS ESPACIALES
POR LA UNIVERSIDAD POLITÉCNICA DE MADRID

POLITÉCNICA

Data handling

Laboratory manual

21 April 2021

Grupo STRAST
Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos
https://www.dit.upm.es/str

Table of contents

Acronyms

ADC	Analog to digital converter
CPU	Central processing unit
GCC	GNU compilation system
GNAT	GNU Ada translator
GNU	GNU is not Unix
GPIO	General purpose input-output
GPS	GNAT Programming Studio
IDE	Integrated development environment
MCU	Microcomputer unit
OBC	On-board computer
OBDH	On-board data handling
OBSW	On-board software
OS	Operating system
PC	Personal computer
RTA	Response-time analysis
TC	Telecommand
™	Telemetry
UART	Universal asynchronous receiver-transmitter
USB	Universal serial bus

Introduction

This document provides instructions for laboratory work in the field of on-board data handling systems. The laboratory is part of the *Data handling course of* the UPM *Master in Space Systems (MUSE)* 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, to be returned at the end of the semester.

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. Simple housekeeping program.
4. Tasking program.
5. Distributed program.
6. Real-time program, including temporal analysis
7. On-board data handling (OBDH) system.

References

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.

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.

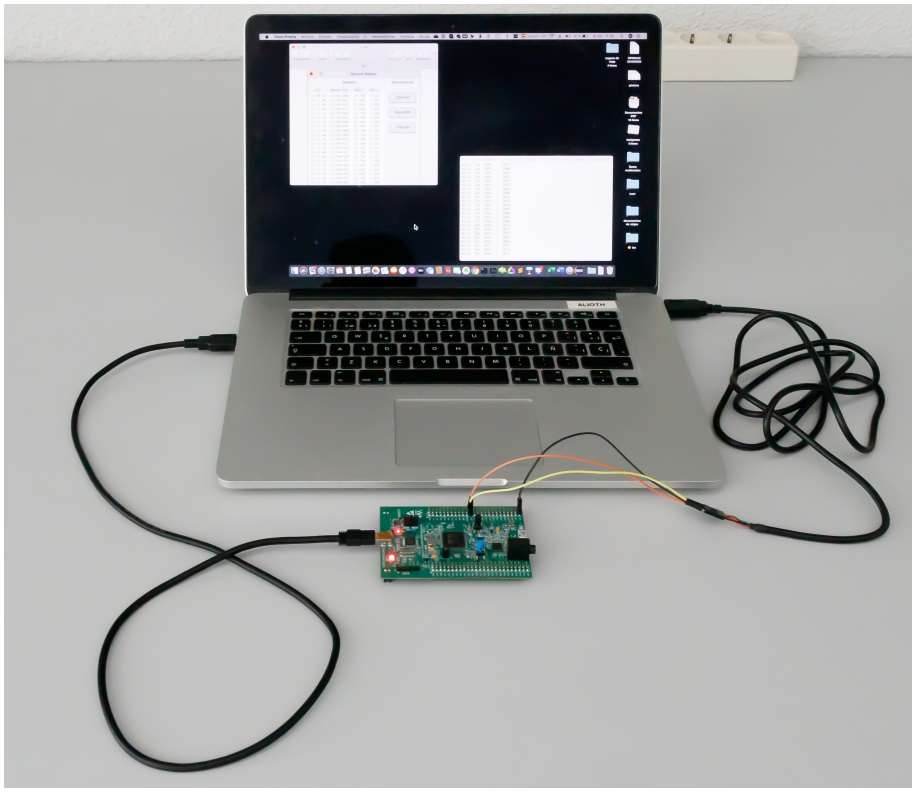

FIGURE 1. LABORATORY KIT.

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.

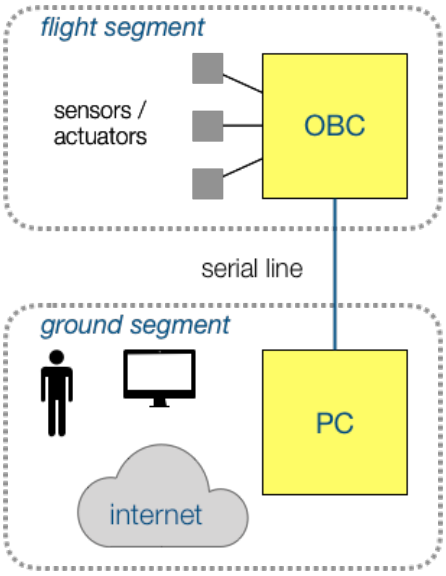

FIGURE 2. ARCHITECTURE OF THE LABORATORY SYSTEM

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 centred on programming the computer board. The ground station software will be provided by the teachers.

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 sensors of a satellite.

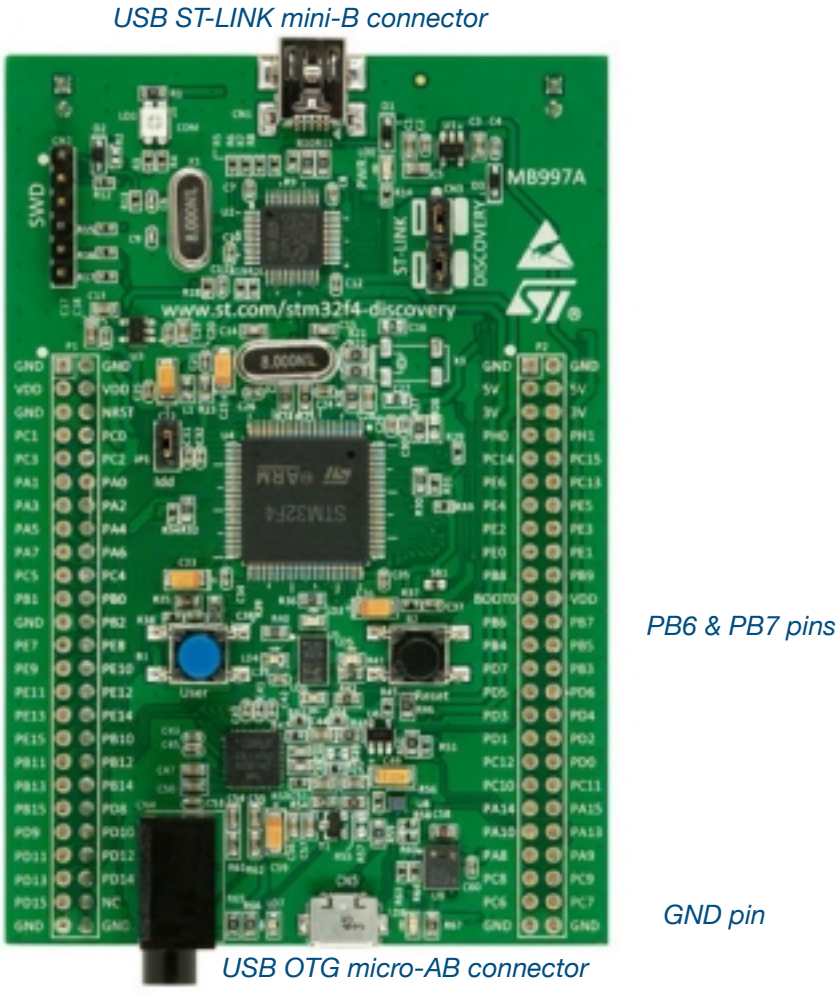

FIGURE 3. COMPUTER BOARD.

Assignment 1 - Install a native programming environment

The aim of this assignment 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,

1.1 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

Download the file

```
gnat-community-2019-20190517-x86_64-windows-bin.exe
```

Run the file and follow the instructions.

MacOS

Download the file

```
gnat-community-2019-20190517-x86_64-darwin-bin.dmg
```

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.

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

Download the file

```
gnat-community-2019-20190517-x86_64-linux-bin
```

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.

1.2 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. Figure 4 shows the main GPS window, which is composed of the following areas:

- 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 figure 4.
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.

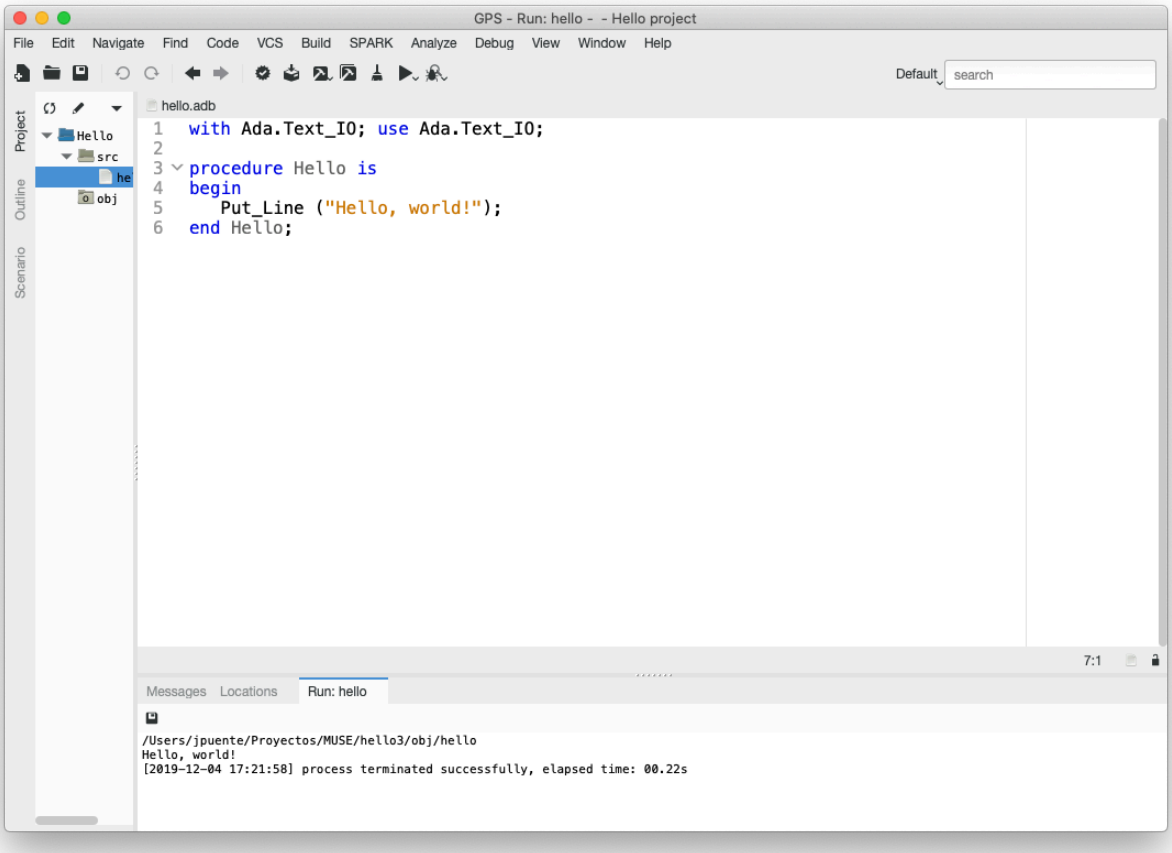

FIGURE 4. GNAT PROGRAMMING STUDIO (GPS)

Assignment 2 - Install the cross-compilation tools

The aim of this assignment 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.

2.1 Cross-compilation tools

The computer board will 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 SMT32 board is the target platform.

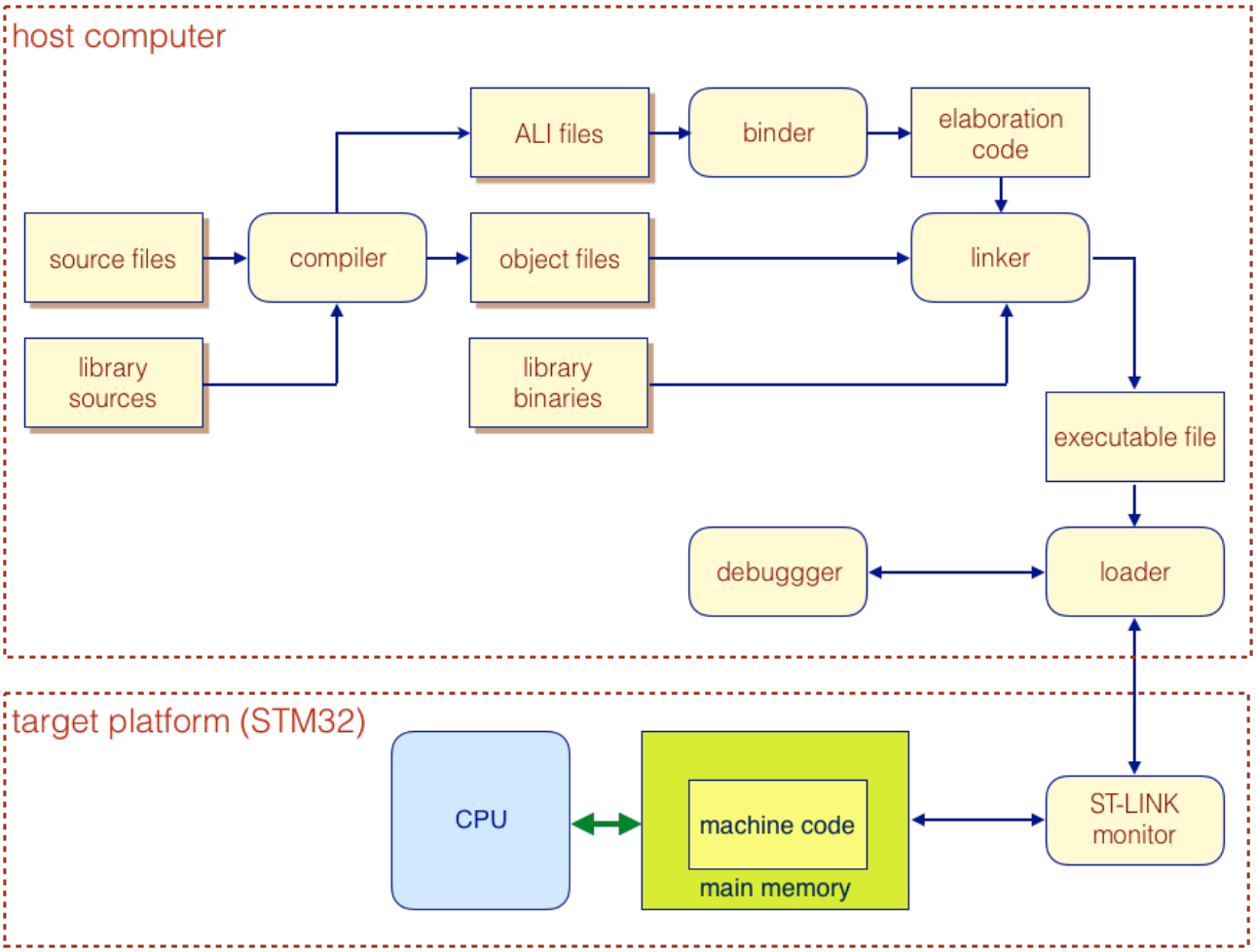

FIGURE 5. CROSS-COMPILATION AND DEBUGGING SYSTEM.

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 pre-installed on the target board that supports loading and debugging from the host platform.

2.2 Download and install GNAT ARM ELF

GNAT ARM ELF is the cross-compilation chain to be used with the STM32F4 board. It can be dowloaded 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

Select ARM ELF (hosted on windows64) and download the file

```
gnat-community-2019-20190517-arm-elf-windows64-bin.exe
```

Run the file and follow the instructions.

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, 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

Download the file

```
gnat-community-2019-20190517-arm-elf-darwin-bin.dmg
```

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.

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

Download the file

```
gnat-community-2019-20190517-arm-elf-linux64-bin
```

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.

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.

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

2.3 Test your installation with an embedded program

The next thing is to compile a 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.

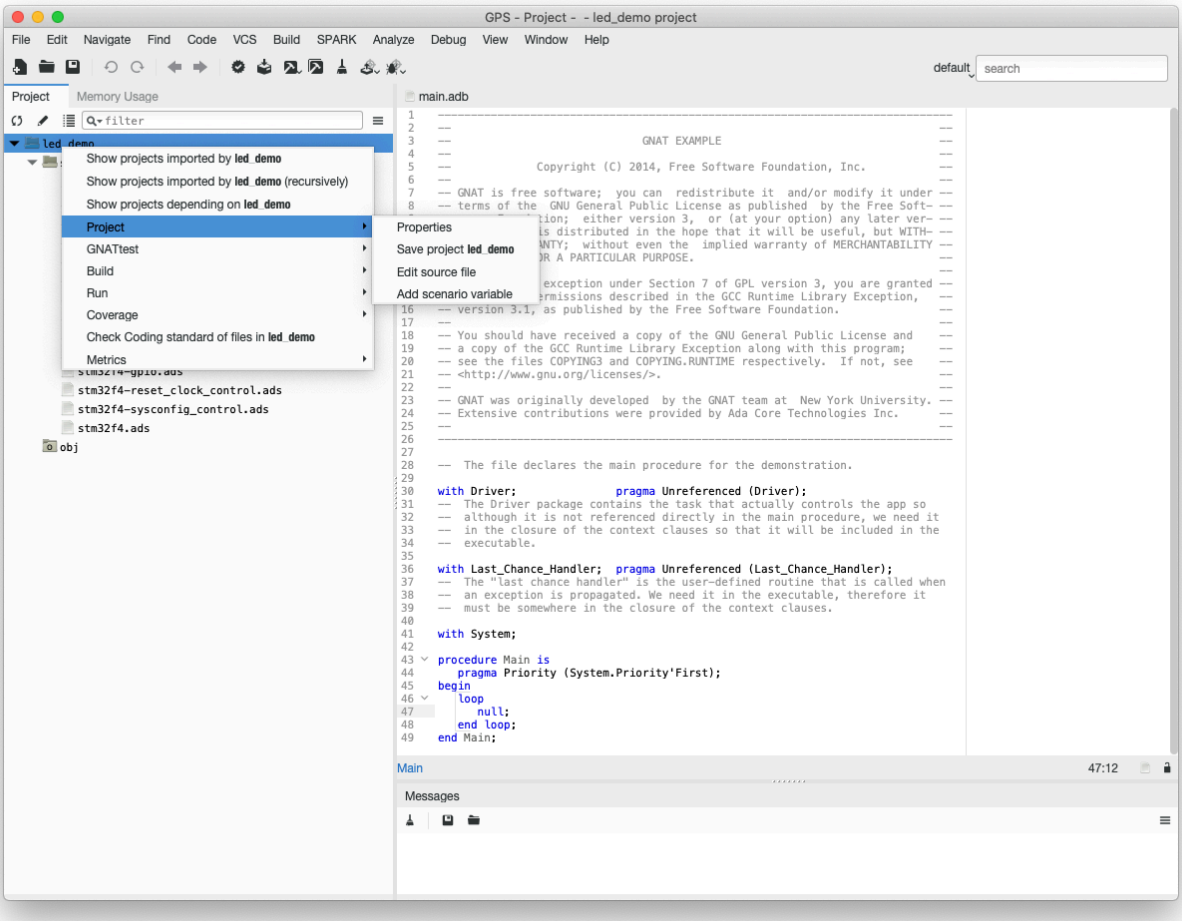

FIGURE 6. CROSS-COMPILATION DEMO PROGRAM

2.4 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. 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.

2.5 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 …/`OBDH_LABS/Ada_Drivers_Library/examples/STMF4_DISCO` and open the `blinky_f4disco.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). When the loading is complete, you will see the board LEDS blinking all at the same time.

Assignment 3 - Simple housekeeping program

The aim of this assignment is to experiment with a simple housekeeping program that only implements a basic function, reading a temperature sensor on the on-board computer. The value read by the sensor is denoted as `OBC_T` (OBC temperature).

The software is organised in modules, in such a way that it can be later extended to a more complex housekeeping system in the next assignments.

3.1 Temperature sensor

The internal temperature sensor in the MCU is used in this assignment. No additional hardware is required.

The STM32F407 reference manual (section 13.10) states that the internal temperature sensor 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 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:

   Temperature (in °C) = {(VSENSE – V25) / Avg_Slope} + 25

   Where:

   - V25 = VSENSE value for 25° C (table 1)
   - Avg_Slope = average slope of the temperature vs. VSENSE curve (table 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 ±1.5 °C. Its main characteristics are described in the STM32F407 datasheet (table 1).

TABLE 1. STM32F407 temperature sensor characteristics.

Symbol	Parameter	Min	Typ	Max	Unit
T_L (1)	V_{SENSE} linearity with temperature	-	±1	±2	°C
Avg_Slope (1)	Average slope	-	2.5		mV/°C
V_{25} (1)	Voltage at 25 °C	-	0.76		V
t_{START} (2)	Startup time	-	6	10	µs
T_{S_temp} (2)	ADC sampling time when reading the temperature (1 °C accuracy)	10	-	-	µs

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

3.2 Software architecture

The software architecture of the simple housekeeping program is depicted in figure 7.[1] The software components are:

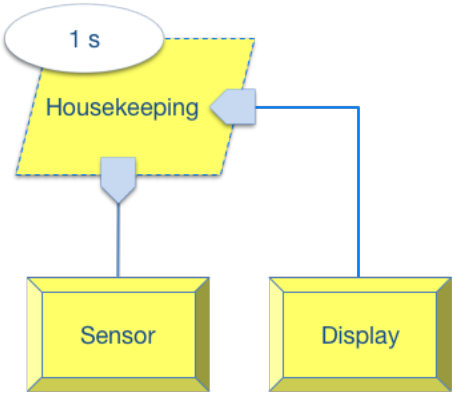

FIGURE 7. SOFTWARE ARCHITECTURE OF SIMPLE HOUSEKEEPING SYSTEM

- `Housekeeping`. Main component, which performs the basic functionality of the system, i.e. reading a temperature value and displaying the value on the console.
- `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.
- `Display`. This module provides a high-level interface to a text console where the measured temperature values can be output.

Since the OBC board does not have a text output device, it has to be 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.

3.3 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. Click on **Clone or download**, download a zip archive, unzip and move to your work directory. The code for this assignment is in the LAB3 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 three subpackages:

- `Housekeeping.Data` contains the definitions of the data types used in the subsystem. Only one data type, `Analog_Data`, is defined for this version of the software.
- `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.Display` includes the procedure `Put`, which is used to display temperature values on the debugger console (see below). The original implementation of this procedure writes raw sensor values, which are integers in the range 0 to 4095, as directly provided by the ADC hardware. These values have to be converted to engineering units. i.e. degrees Celsius, using the steps shown in section 3.1 above. The software provided to the students includes a program, `adc2celsius`, which implements this functionality.

  The `Display.Put` procedure uses the `Ada.Tex_IO` package to write to the standard output. Since there is no device that can be used to provide text output on the OBC board, the ST-LINK prove provides a facility, which is called *semihosting*, to provide this functionality. When the

[1] The graphic notation is AADL (Architecture Analysis and Design Language).

program is run using the cross-debugger on the host, the board standard output is redirected to the debugger console. Therefore, in order to see the temperature values the program must be run from the debugger (see below).

The main procedure is `OBSW`[2]. It calls `Housekeeping.Initialize`, which initializes the sensor and then calls the `Run` procedure. This procedure executes and endless loop that performs the following actions:

- Get a raw temperature measurement from the sensor
- Display the value

Additionally, one of the board 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.

3.4 Compile and run with the debugger

Open GPS and do the following:

1. Select **Open project** on the welcome window. Navigate to the `LAB3` directory and open the `simple_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 flash memory. After that, the debugger is started,[3] and the debugger console shows the following lines:
   ```
   (gdb) monitor reset halt
   (gdb)
   ```
3. Type the following command on the debugger console:
   ```
   (gdb) continue
   ```
4. The program will start running (check the LED blinking), and the raw temperature readings are shown on the Messages tab of the debugger console (figure 8).

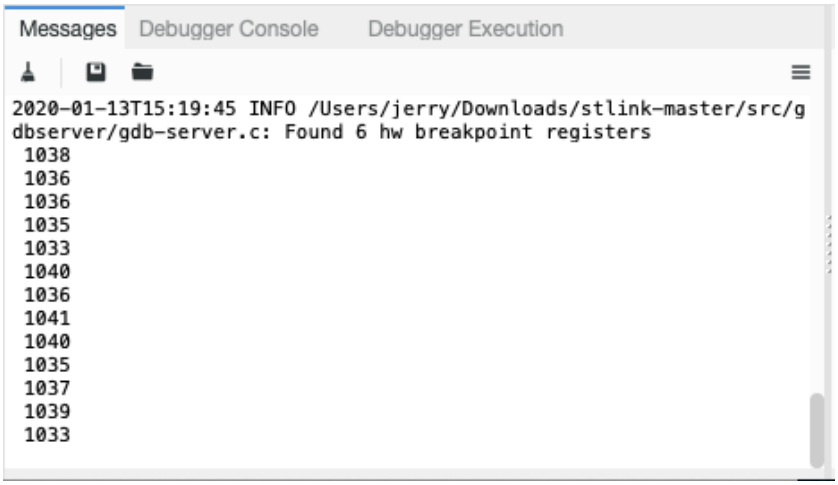

FIGURE 8. DEBUGGER OUPUT

The raw measurement values can be converted to Celsius using the `adc2celsius` program. You should take into account that the internal temperature sensor does not provide an accurate measurement, and may have an offset that varies from one chip to another.

---

[2] On-Board Software

[3] On Windows a message will be displayed requesting permission to connect st-util to external networks. Be sure to grant such permission to enable the debugger connection to the board.

3.5 Make changes to the program

As a final activity, you may make some changes to the provided program in order to make sure that you understand the logics behind the source code. Proposed changes are:

- 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);
  ```

  The effect of this statement is to delay the execution of the program fo 1 s. You will have to import the `Ada.Real_Time` library package in order to use the operations included in the statement.

Assignment 4 - Tasking housekeeping program

The aim of this assignment is to extend the simple housekeeping program of the previous assignment by adding a communications subsystem. The extended system includes two concurrent tasks communicating through a protected shared object.

4.1 Software architecture

The software architecture of the tasking housekeeping program is depicted in figure 9. The software components are:

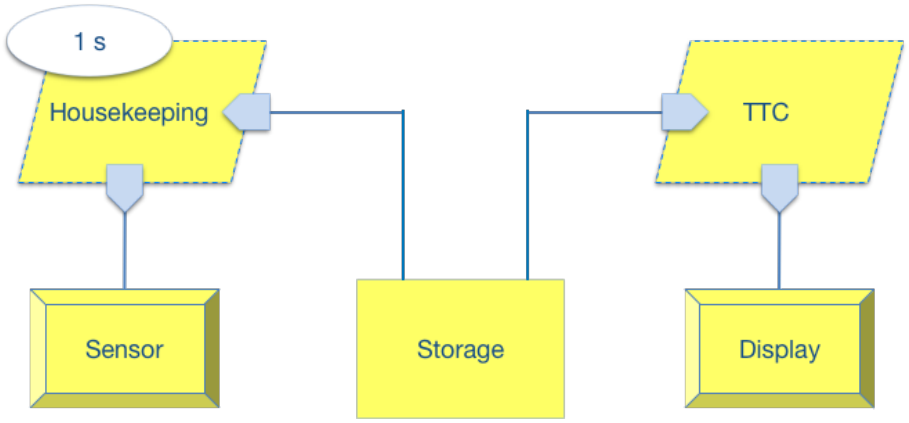

FIGURE 9. SOFTWARE ARCHITECTURE OF TASKING HOUSEKEEPING SYSTEM

The differences with the previous architecture are (figure 9):

- There is a new component, TTC, that handles the display.
- Both the `Housekeeping and TTC` components include concurrent tasks.
- The `Housekeeping and TTC` tasks communicate through a new component, `Storage`. This component is a data object storing one temperature value, which is written by `Housekeeping` and read by `TTC`.

4.2 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. Click on **Clone or download**, download a zip archive, unzip and move to your work directory. The code for this assignment is in the LAB4 folder.

As in the previous assignment, the `Housekeeping` package is the root element of the housekeeping subsystem. Its specification and body is similar to the previous version, except that it now contains a concurrent task, `Housekeeping_Task`, and the values read from the sensor are sent to `Storage` instead of `Display`. This package has been moved to the `TTC` subsystem, but otherwise remains similar.

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 puts them on the display.

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 OBSW main procedure initialises the board LEDs and the `Housekeeping` and `TTC` subsystems. The activity if both subsystems is carried out by their respective tasks, which start executing concurrently with the main task. The initialization procedures return to the main procedure, which enters and endless loop doing nothing and running in parallel with the other tasks. In order not to disturb the execution of the subsystems tasks, the main loop runs at the lower possible priority, which is specified in the `obsw.ads` file.

4.3 Compile and run with the debugger

As in the previous project, open GPS and do the following:

1. Select **Open project** on the welcome window. Navigate to the `LAB4` directory and open the tasking_`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 flash memory. After that, the debugger is started, and the debugger console shows the following lines:
   ```
   (gdb) monitor reset halt
   (gdb)
   ```
3. Type the following command on the debugger console:
   ```
   (gdb) continue
   ```
4. The program will start running (check the LED blinking), and the raw temperature readings are shown on the Messages tab of the debugger console like in the previous project.

4.4 Make changes to the program

You may include the same changes that were proposed in the previous assignment:

- Include the conversion to Celsius in the `Display.Put` procedure.

Assignment 5 - Distributed housekeeping program

The two previous versions of the housekeeping program display the measured values on a debugger console. This means that the program must be run from the debugger, with the ST-LINK cable in place.

A more realistic solution uses a serial interface to send these values to a simulated ground station running on the host computer, as shown in figure 2. The aim is to simulate the radio link between the satellite and the ground station.

5.1 Hardware connections.

This scheme makes use of the USB/UART interface cable provided to the students. The USB/UART cable has a TTL conector that must be connected to the STM32f4 board pins that convey the serial line (UART) signals (figure 10).

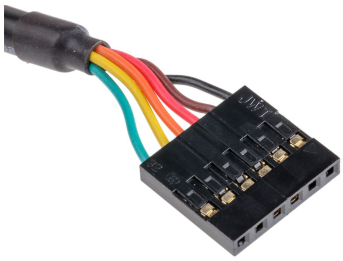

FIGURE 10. UART CABLE CONNECTOR

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

TABLE 2. SERIAL LINE CONNECTIONS ON BOARD.

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

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 and a transmission rate of 115200 bps with parity 8N1.

5.2 Host terminal application

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 11).

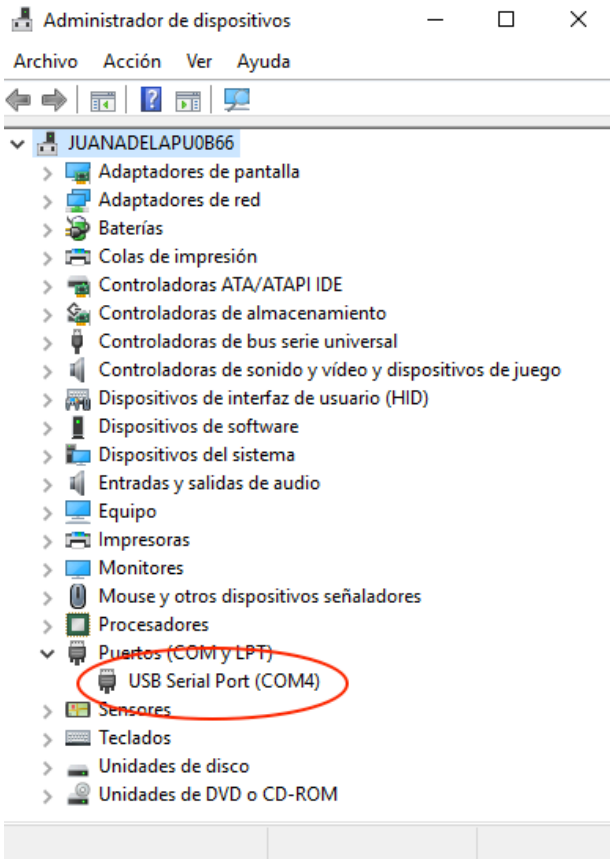

FIGURE 11. IDENTIFICATION OF USB SERIAL PORT

Now, to set up PuTTY, open the application and set the configuration parameters as shown in figure 12.

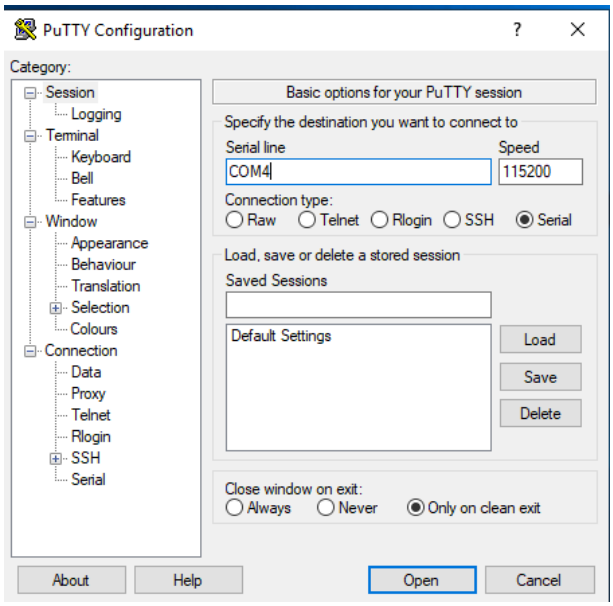

FIGURE 12. PUTTY CONFIGURATION

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
```

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`.

Linux

The recommended application is `screen`,[4] 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`.

5.3 Software architecture

The software architecture is similar to the previous project, except that the display is replaced by a serial line handler adapted from the examples in the Ada Drivers Library.

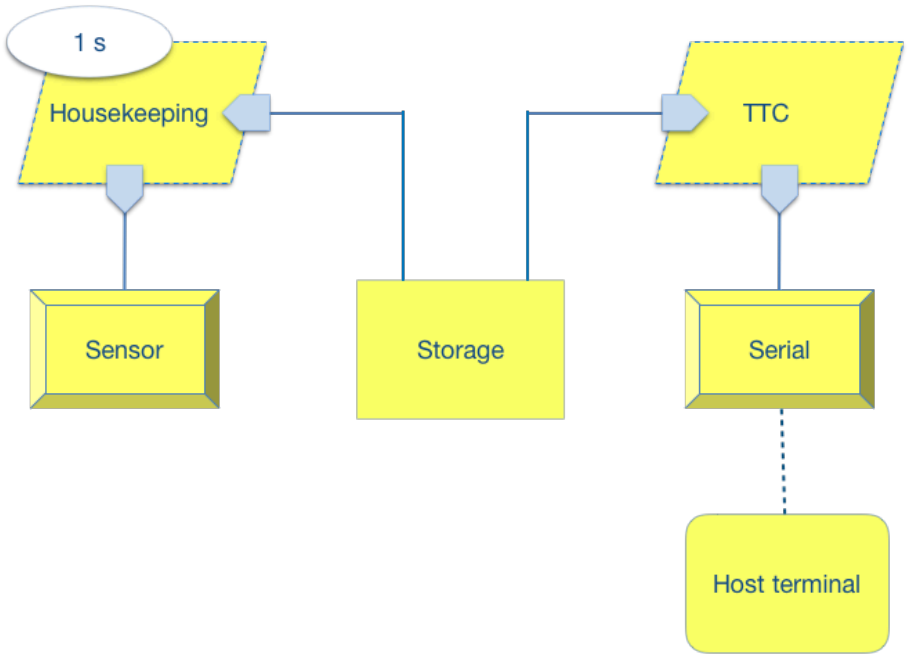

FIGURE 13. SOFTWARE ARCHITECTURE OF DISTRIBUTED HOUSEKEEPING SYSTEM

[4] `gtkterm` is a good alternative.

5.4 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. Click on Clone or download, download a zip archive, unzip and move to your work directory. The code for this assignment is in the LAB5 folder.

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 rest of the implementation is the same as in the previous project.

5.5 Compile and run

This time there is no need to run the program from the debugger. Open GPS and do the following:

1. Select Open project on the welcome window. Navigate to the LAB5 directory and open the `distributed_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 5.2. The sensor measured values will start being displayed on the host application.

5.6 Make changes to the program

You may include changes similar to the ones proposed in the previous assignments:

- Include the conversion to Celsius in the `Display.Put` procedure.

Assignment 6 - Real-time program

The next version of the housekeeping program includes real-time requirements and the use of a real-time clock to add a timestamp to the housekeeping data sent to the ground station, simulated by the serial connection to the host PC like in the previous assignment. The hardware connections and the use of a host terminal application remain the same.

6.1 Software architecture

The software architecture now includes a period of 10 s for the TTC task. This task reads the last value from the storage every 10 s (figure 14).

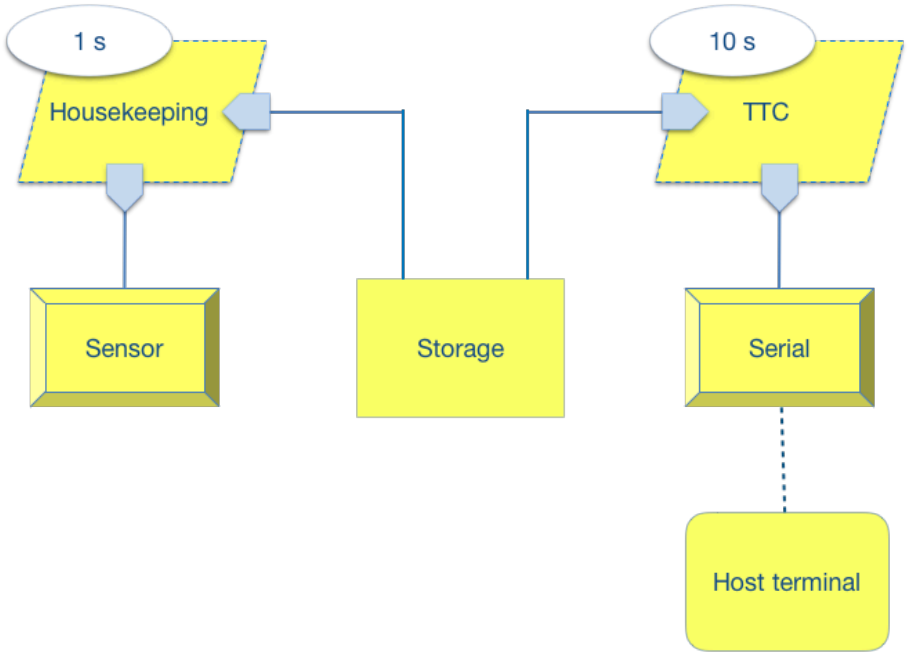

FIGURE 14. SOFTWARE ARCHITECTURE OF REAL-TIME HOUSEKEEPING SYSTEM

6.2 Real-time requirements

The following real-time requirements are specified for the system:

- The `Housekeeping` task executes with a period of 1 s and has a deadline equal to its period, (1 s).
- The `TTC` task executes with a period of 10 s and has a deadline of 2 s.

Priorities are assigned in deadline-monotonic order, as shown in table 3. The `Buffer` protected object, which is part of the `Storage` implementation, is accessed by both application tasks and thus has a ceiling priority equal to the priority of the `Housekeeping` task.

TABLE 3. REAL-TIME REQUIREMENTS

Task	Period	Deadline	Priority
Housekeeping	1.0	1.0	20
TTC	10.0	2.0	10
Storage Buffer	—	—	20

6.3 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. 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 implementation code differs from the previous project in several aspects.

- Period and priority values have been explicitly added to the specification of the `Housekeeping` and `TTC` packages. A start delay has been added to the respective tasks, in order to let all the packages initialize before the regular operation of the system starts.
- The ceiling priority of the `Storage` buffer has been set to the same value as the `Housekeeping` task.
- A new `State` data type has been defined, which is a record including a timestamp and an analog data value. Messages sent to ground are now of this data type.

  Timestamps are refined as 64-bit integers, denoting the number of second elapsed since the beginning of the mission. To the purpose of this laboratory this value is taken from the real-time clock provided by the `Ada.Real_Time` library package.
- In order to improve the visual aspect of the messages as viewed on the host terminal application, a new package, `Data_Images`, has been added that provides fixed-width string images of mission time and analog data values.

The rest of the implementation is the same as in the previous project.

6.4 Compile and run

Open GPS and do the following:

1. Select Open project on the welcome window. Navigate to the LAB6 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 5.2. The sensor measured values together with their respective timestamps will start being displayed on the host application (figure 15).

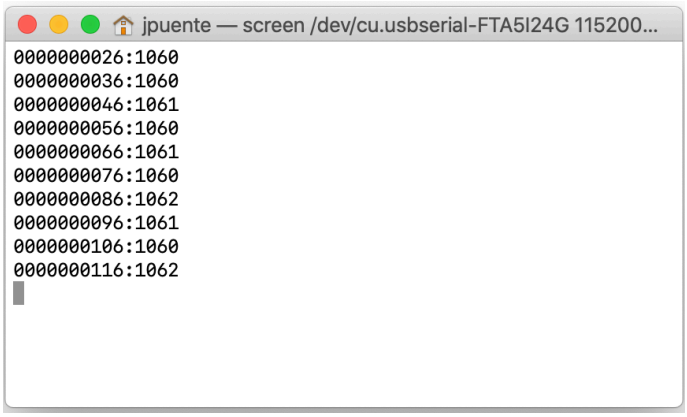

FIGURE 15. SAMPLE OUTPUT ON HOST TERMINAL

6.5 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 16). 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.

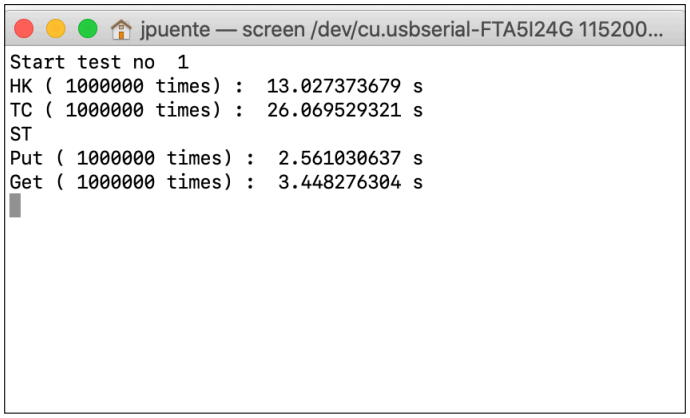

FIGURE 16. OUTPUT OF WCET MEASUREMENT TOOL

In the example shown on figure 16, the HK execution time has been measured 1 000 000 times, with a total measurement time of 13.02 s. Therefore, the value to be taken for the response time analysis is $13.02 \cdot 10^{-6}$ s = 13.02 µs, 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 4.

TABLE 4. DATA ARRANGEMENT FOR RTA OF THE HOUSEKEEPING SYSTEM.

Task	T	C	B	D	R	P	C_{Put}	C_{Get}
Housekeeping	1.0	$13 \cdot 10^{-6}$	$4 \cdot 10^{-6}$	1.0	$17 \cdot 10^{-6}$	20	$3 \cdot 10^{-6}$	—
TTC	10.0	$26 \cdot 10^{-6}$	0	2.0	$39 \cdot 10^{-6}$	10	—	$4 \cdot 10^{-6}$
						CP	**20**	10

Final project - OBDH system

The final version of the housekeeping program is a full OBDH system, including an additional sensor reading and the reception and interpretation of elementary telecommands from the ground station.

The ground station is implemented by a separate program running on the host PC platform. The radio connection between the OBDH software running on the OBC board and the ground station running on the host PC is simulated by a serial cable connection, as in the previous assignments.

7.1 Software architecture and functional overview

The software architecture is shown on figure 17. The system consists of four subsystems, very much like the ones found in a real on-board satellite system.

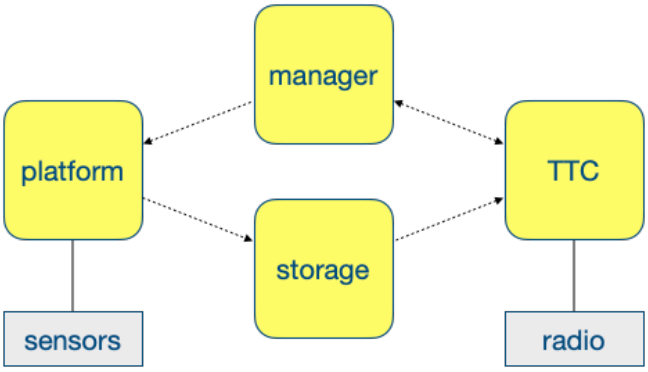

FIGURE 17. OBDH SYSTEM ARCHITECTURE

- The **platform** subsystem performs housekeeping functions on the satellite platform. It is expanded from the housekeeping component developed in the previous laboratory assignments in order to include an additional sensor. The list of variables that are monitored is now:
  - OBC_T : OBC temperature
  - OBC_V : OBC voltage

  The *state* of the platform is the set of values measured at a particular time, with a *timestamp* indicating the time at which they have been acquired. *Mission time*, a monotonic seconds count from the system start time, is used to this purpose.
- The **storage** subsystem keeps trace of the last *N* state values measured by the platform subsystem, where *N* is a configurable system parameter.
- The **TTC** system is in charge of all communications with the ground station. Its functionality includes:
  - **Telemetry** (TM) messages transmitted to ground, which may be of the following kinds:
    - Basic telemetry (*Hello*) messages, including the last measured values from all the sensors. These messages are periodically transmitted when the system is in *idle* mode (see below).
    - *Housekeeping* messages include a more complete record with the last *N* stored values of the state. These messages are transmitted in response to a telecommand.
    - *Mode* messages indicating the current operating mode of the system are transmitted after every mode change (see below).
    - *Error* messages are occasionally sent to indicate some kinds of errors.
  - **Telecommands** (TC) are messages received from ground, and can be of the following kinds:
    - *Open_Link* messages are sent from the ground station in order to start a coverage period (see below).
    - *Request_HK* telecommands are used to request the OBDH system to send a *housekeeping* telemetry message.

- *Close_Link* telecommands are sent by the ground station in order to close a coverage period and return to the *idle* mode (see below).
- An *error* TC value is signalled by the TTC subsystem when a message received from ground cannot be properly decoded as a valid TC.

- The **manager** subsystem carries out functions related to the operating mode of the system and the execution of telecommands. In this simplified OBDH system only two modes of operation are defined, related to the (simulated) visibility of the satellite from the ground station.
  - **Idle**. The system is this mode when the satellite is not visible from the ground station.
  - **Coverage**. The system is in coverage mode when the satellite is visible from the ground station.

  Mode changes are started from the TTC subsystem, according to the following protocol:
- When in **idle** mode, the OBDH system periodically transmits basic TM messages, and listens to telecommands from ground. When an *open_link* TC is received, it requests the manager to switch to the **coverage** mode. No other kinds of telecommands are accepted in this mode.
- When in **coverage** mode, basic telemetry is not transmitted, and the TTC subsystem listens to Telecommands from the ground station. The system switches back to idle mode when a *close link* TC is received or, alternatively, a maximum coverage time span has passed.

7.2 System design

The task structure of the system that has been designed in order to provide the above functionality is shown on figure 18.

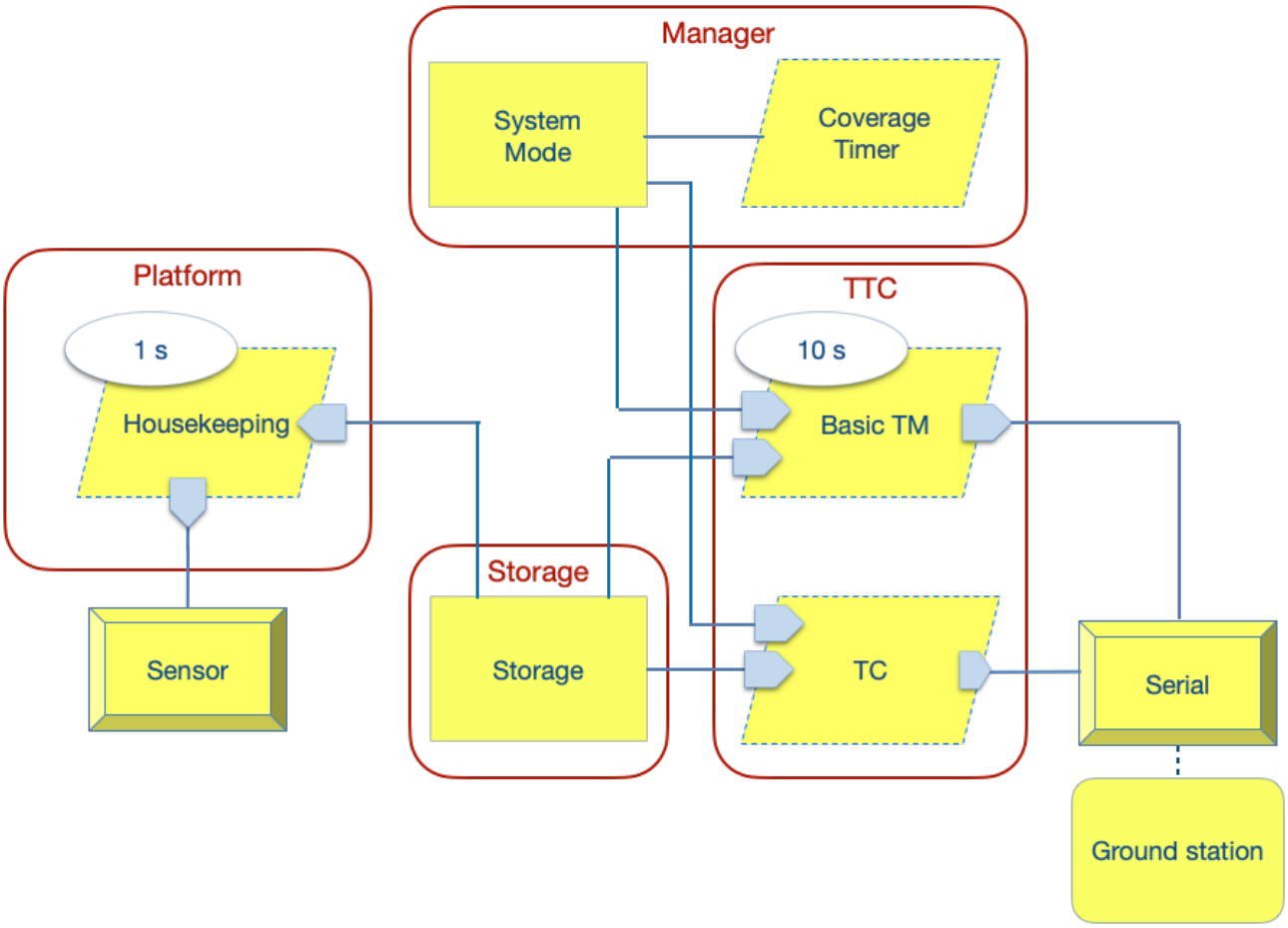

FIGURE 18. OBDH SYSTEM TASK STRUCTURE

7.3 Real-time requirements

The following real-time requirements are specified for the system:

- The `Housekeeping` task executes with a period of 1 s and has a deadline of 100 ms.
- The `Basic_TM` task executes with a period of 10 s and has a deadline of 500 ms.
- The TC task is sporadic, and is executed upon reception of a telecommand. The minimum separation of the event is 2 s, and the deadline is 1 s.
- The Coverage_Timer task is sporadic with a minimum separation of 60 s and a deadline of 1 s.

Priorities are assigned in deadline-monotonic order, as shown in table 6. The `Buffer` protected object, which is part of the `Storage` implementation, is accessed by the `Housekeeping`, `Basic_TM` and `TC` tasks, and thus has a ceiling priority equal to the priority of the `Housekeeping` task.

TABLE 5. OBDH REAL-TIME REQUIREMENTS

Task	Period	Deadline	Priority
Housekeeping	1.0	0.1	20
Coverage_Timer	60.0	1.0	15
Basic_TM	10.0	0.5	12
TC	1.0	1.0	10
Storage Buffer	—	—	20

7.4 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. Click on **Clone or download**, download a zip archive, unzip and move to your work directory. The code for the OBDH system is in the `PROJECT/OBDH` folder.

The implementation code reflects the task structure in figure 18.

7.5 Compile and run

Open GPS and do the following:

1. Select **Open project** on the welcome window. Navigate to the `PROJECT/OBDH` directory and open the obdh`.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, following the instructions in section 5.1 of this manual.
4. Identify the serial port name on the host computer and launch the remote terminal application as explained in section 5.2.
   The output shows all telemetry messages received from the board, including basic housekeeping in idle mode and responses to telecommands (figure 19).

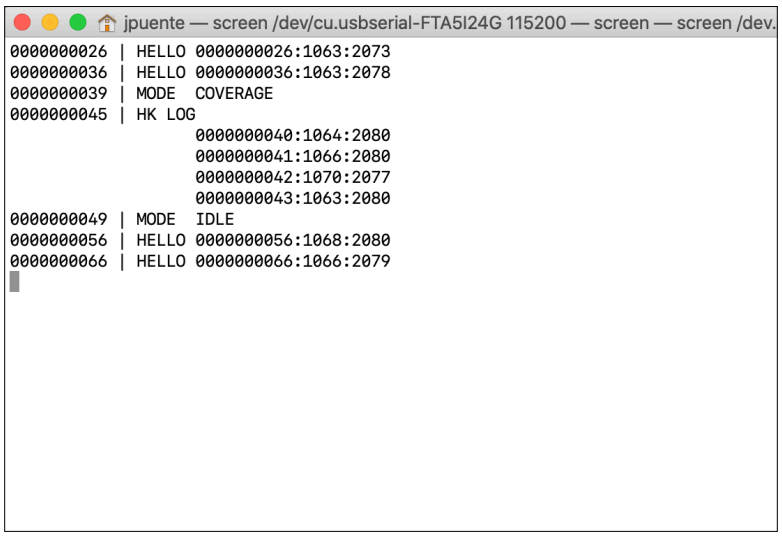

FIGURE 19. SAMPLE TELEMETRY MESSAGES RECEIVED AT THE HOST TERMINAL.

7.6 Ground station

The appearance of the output can be improved by using dedicated software. A simple example is the python script gs.py provided with the OBDH software. In order to use it, do the following:

1. Install Python in your system, if not already installed.
2. Install the `pip` package manager if not already installed
3. Install the `pySerial` module:

   ```
   python  -m pip install pyserial
   ```

4. Check the serial port name in the `gs.py` script and change it according to your local setup, which can be found as explained in section 5.2.
5. Run the script from a terminal window, with the board connected to the host PC:

   ```
   python  gs.py
   ```

A sample of the telemetry messages received at the ground station is shown in figure 20.

Telecommands must be entered entered through the keyboard of the host computer where the ground station is running. The telecommands listed in section 7.1 above are abbreviated as follows:

- `open`: *Open_Link*.
- `request`: *Request_HK*.
- `close`: *Close_Link*.

The `exit` command terminates the execution of the ground station.

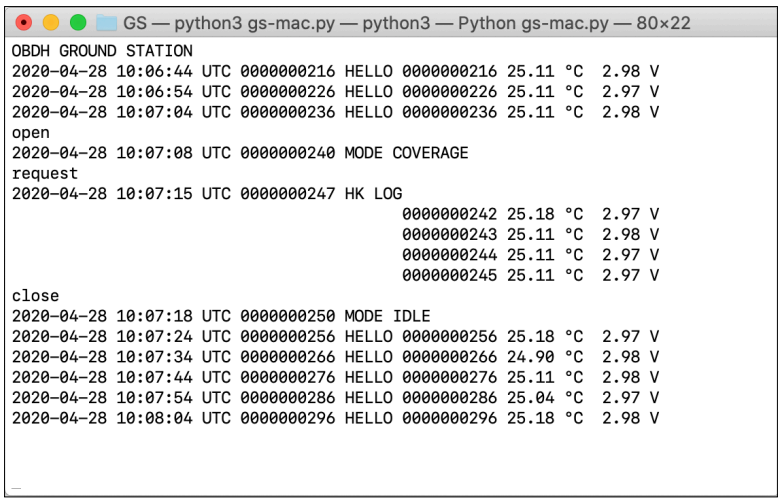

FIGURE 20. SAMPLE TELEMETRY MESSAGES RECEIVED BY THE GS SCRIPT. TELECOMMANDS ENTERED THROUGH THE KEYBOARD ARE ALSO SHOWN.