PROJECT 1 (PART A)

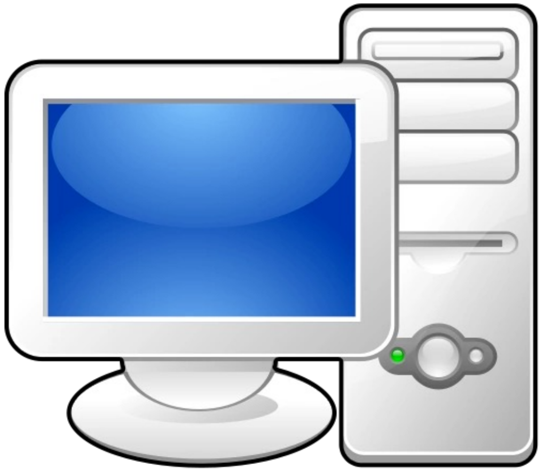
Onboard Spacecraft Software

Ojective

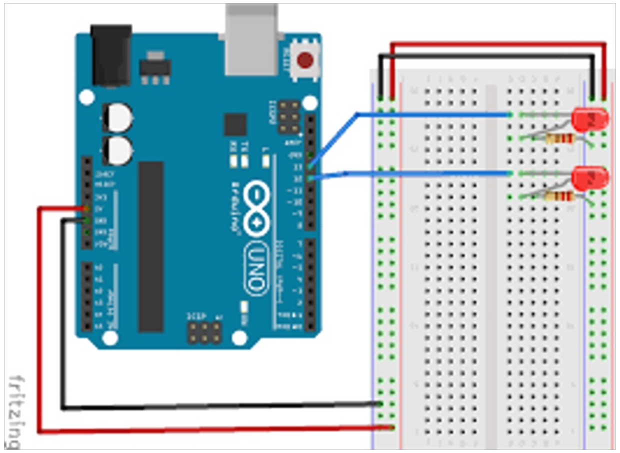
- To build a prototype for a reduced onboard softward. The system has two main components:
    1. A main computing component to controls the whole system.
    2. A microcontroller subsystem that has a direct access to the sensors and actuators.
- Both subsystems communicate using a master/slave message protocol defined for this project..

Development modules

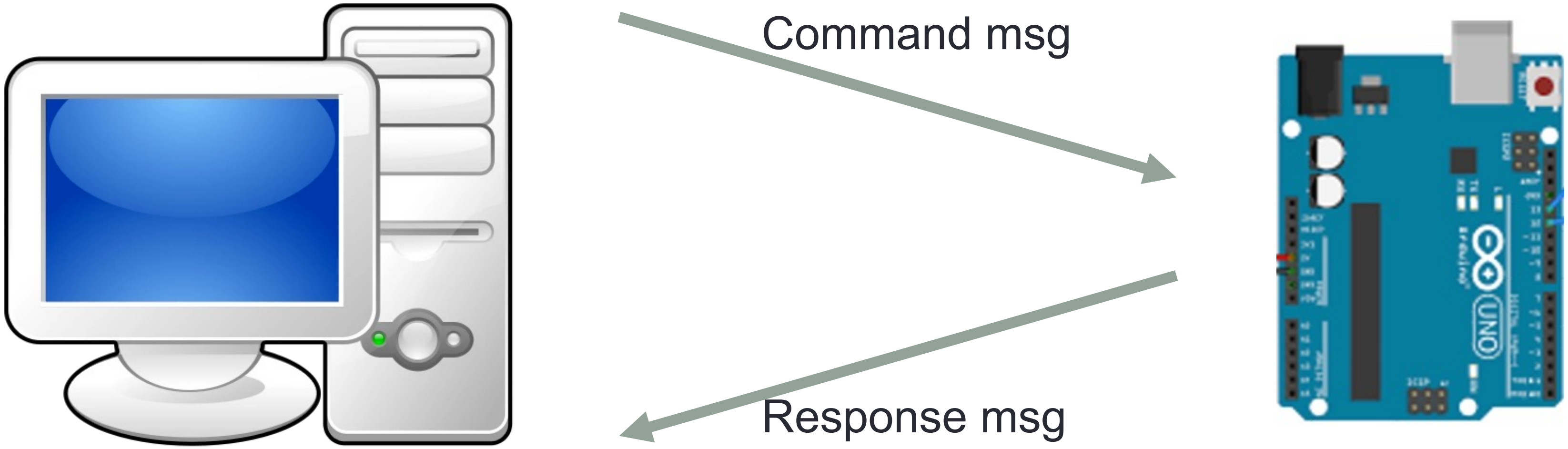
- Software module
  - Desktop PC
  - RTMS O.S.
  - Controls/simulates the logic of the project.



- Hardware module
  - Electornic circuit.
  - Based on Arduino
  - Controls/simulates the hardware of the project.



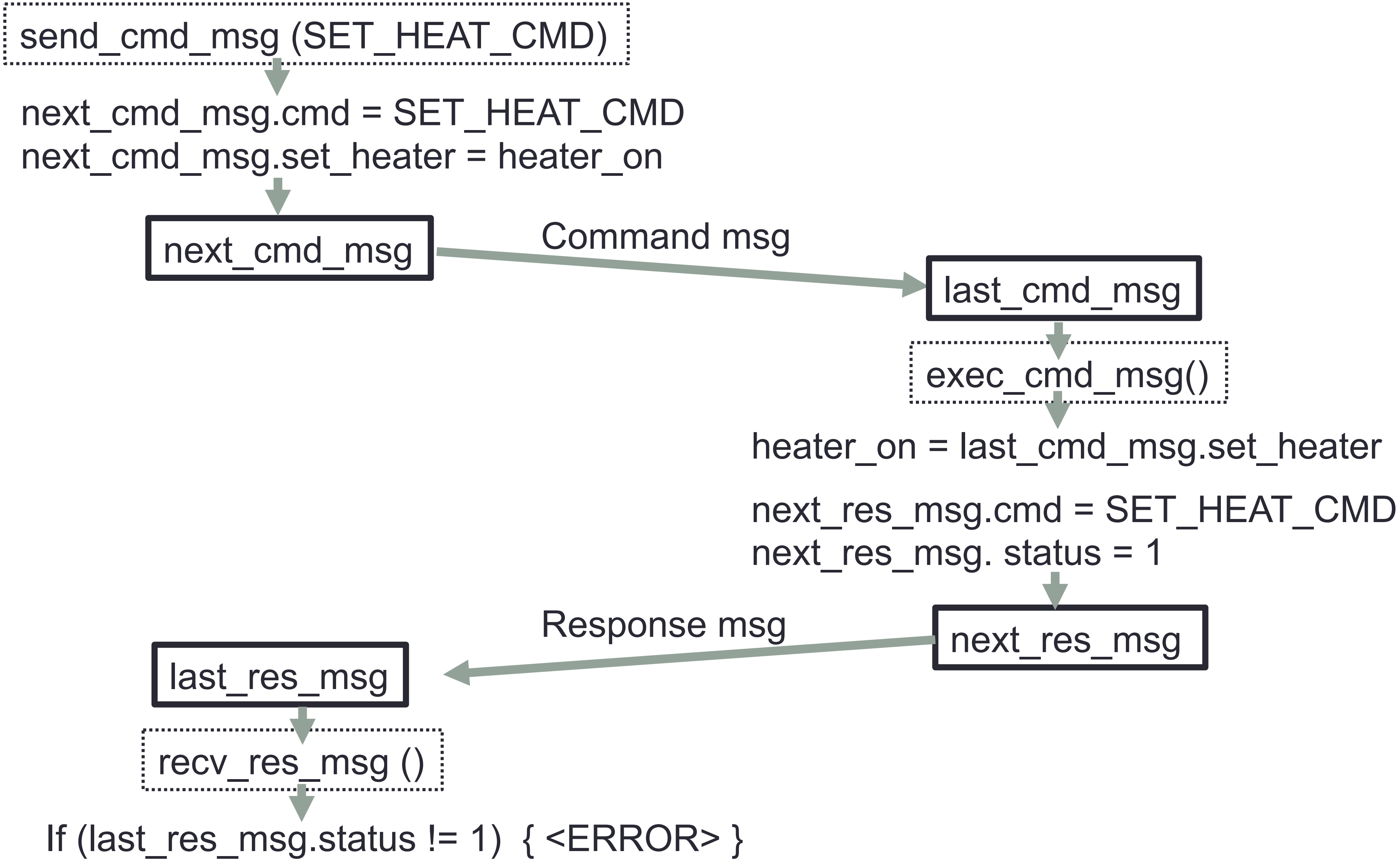
Comunication protocol: Master / Slave



Command msg

Response msg

Protocol example: Actuator



send_cmd_msg (SET_HEAT_CMD)

next_cmd_msg.cmd = SET_HEAT_CMD
next_cmd_msg.set_heater = heater_on

next_cmd_msg

Command msg

last_cmd_msg

exec_cmd_msg()

heater_on = last_cmd_msg.set_heater

next_res_msg.cmd = SET_HEAT_CMD
next_res_msg. status = 1

next_res_msg

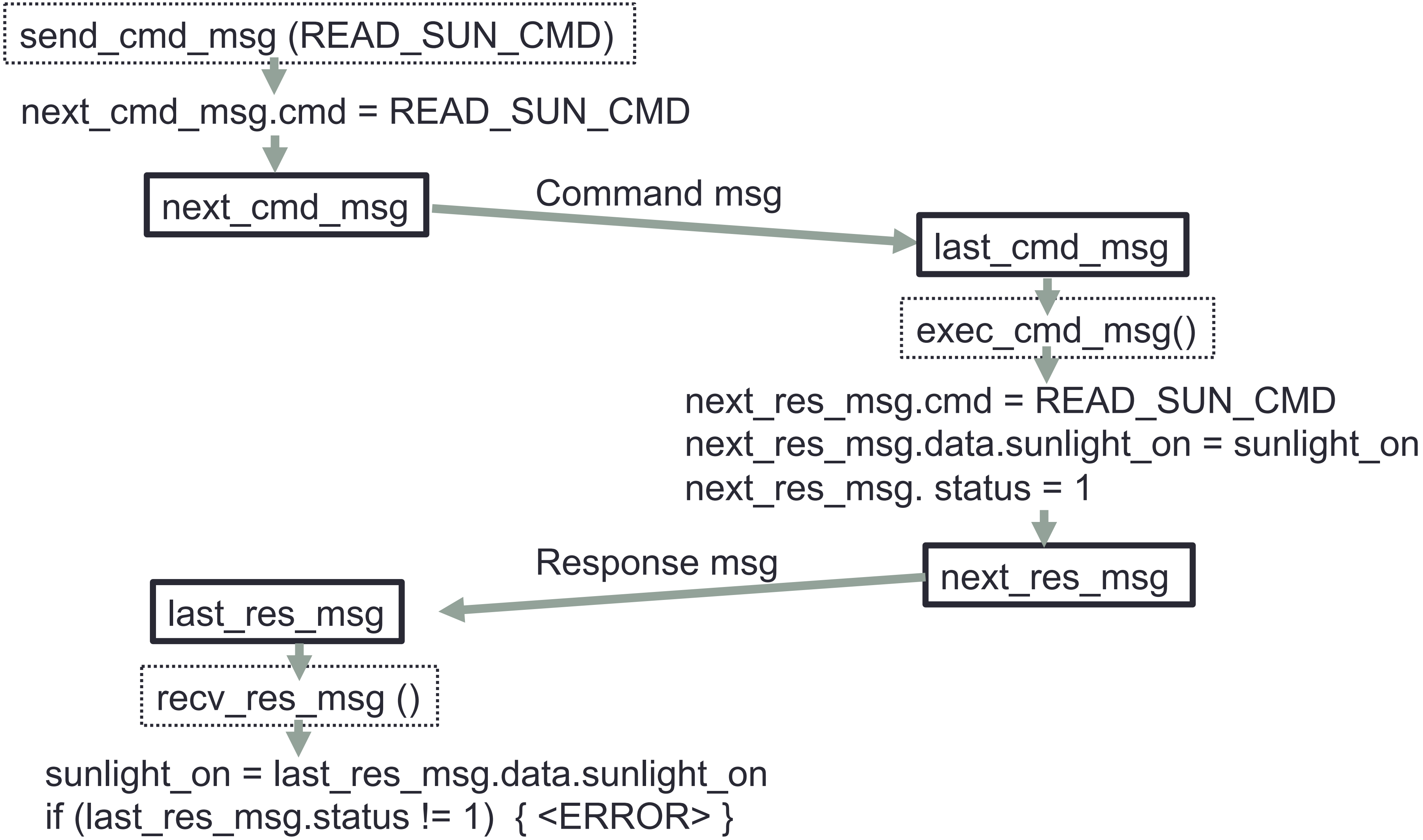
Response msg

last_res_msg

recv_res_msg ()

If (last_res_msg.status != 1) { <ERROR> }

Protocol example: Sensor



send_cmd_msg (READ_SUN_CMD)

next_cmd_msg.cmd = READ_SUN_CMD

next_cmd_msg

Command msg

last_cmd_msg

exec_cmd_msg()

next_res_msg.cmd = READ_SUN_CMD
next_res_msg.data.sunlight_on = sunlight_on
next_res_msg. status = 1

next_res_msg

Response msg

last_res_msg

recv_res_msg ()

sunlight_on = last_res_msg.data.sunlight_on
if (last_res_msg.status != 1) { <ERROR> }

Protocol messages: Commands and responses

Message name	Command data	Response data
NO_CMD	-----	-----
SET_HEAT_CMD	set_heater	status
READ_SUN_CMD	-----	data.sunlight_on status
READ_TEMP_CMD	-----	data.temperature status
READ_POS_CMD	-----	data.position status

Functions interface: Arduino

- Function: get_temperatura()
- Inputs variables taken from the global state:
  1. **heater_on**
  2. **sunlight_on**
  3. **temperature**
  4. **time_temperatura**
- Outputs variables modified on the global state:
  1. **temperature**
  2. **time_temperatura**
- Constants values used (from #define):
  1. **SHIP_SPECIFIC_HEAT**
  2. **SHIP_MASS**
  3. **HEATER_POWER**
  4. **SUNLIGHT_POWER**
  5. **HEAT_POWER_LOSS**

Functions interface: Arduino

- Function: get_position ()
- Inputs variables taken from the global state:
  1. **init_time_orbit**
- Outputs variables modified on the global state:
  1. **position**
- Constants values used (from #define and static global variables):
  1. **ORBIT_POINTS_SIZE**
  2. **ORBIT_TIME**
  3. **orbit_points**

Functions interface: Arduino

- Function: exec_cmd_msg ()
- Inputs variables taken from the global state:
  1. **last_cmd_msg**
  2. **sunlight_on**
  3. **temperature**
  4. **position**
- Outputs variables modified on the global state:
  1. **next_res_msg**
  2. **heater_on**
- Constants values used (from *enum command*):
  1. **NO_CMD**
  2. **SET_HEAT_CMD**
  3. **READ_SUN_CMD**
  4. **READ_TEMP_CMD**
  5. **READ_POS_CMD**

Functions interface: i386

- Function: control_temperature ()
- Inputs variables taken from the global state:
  1. **temperature**
- Outputs variables modified on the global state:
  1. **heater_on**
- Constants values used (from #define):
  1. **MAX_TEMPERATURE**
  2. **MIN_TEMPERATURE**
  3. **AVG_TEMPERATURE**

Functions interface: i386

- Function: send_cmd_msg (enum command cmd)
- Inputs variables taken from the global state:
  1. **heater_on**
- Outputs variables modified on the global state:
  1. **next_cmd_msg**
- Constants values used (from *enum command*):
  1. **NO_CMD**
  2. **SET_HEAT_CMD**
  3. **READ_SUN_CMD**
  4. **READ_TEMP_CMD**
  5. **READ_POS_CMD**

Functions interface: i386

- Function: recv_res_msg ()
- Inputs variables taken from the global state:
  1. **last_res_msg**
- Outputs variables modified on the global state:
  1. **sunlight_on**
  2. **temperature**
  3. **position**
- Constants values used (from *enum command*):
  1. **NO_CMD**
  2. **SET_HEAT_CMD**
  3. **READ_SUN_CMD**
  4. **READ_TEMP_CMD**
  5. **READ_POS_CMD**

How to make a unit test

- The steps to create a unit test for a function are the following:
    1. Set a specific value of your choosing for all the variables that belong to the input of this function.
    2. Call the function.
    3. Check if all the variables that belong to the output of this function have the expected value
        - These expected values have to be previously calculated.

Test example: get_temperature

1. Set the value you want for:
   - **heater_on**, **sunlight_on**, and **temperature**
2. Set time_temperature to the following value:
   - **time_temperature** = **get_clock**() – **elapsed_time**
3. Compute by hand the new temperature
   - Using **elapsed_time** and the values on step 1
4. Check that the new values are correct.
   - Both **temperature** and **time_temperature**.
   - It should be considered a certain margin of error.

Test example: get_position

1. Set **init_time_orbit** to the following value:
   - **init_time_orbit** = **get_clock**() – **elapsed_time**
2. Compute by hand the new position
   - Using **elapsed_time** as the time since the first orbit started.
3. Check that the new position is correct.
   - It should be considered a certain margin of error.