IMU Exercise: Requirements and Design

When developing flight software and using F´ it is important to start the development of a component by laying out the requirements of that component and generating a design of that component. In this portion of the exercise we will walk through both of these steps.

F´ formalizes the design of the component using the FPP modeling language. This allows us to take the design (called a model in FPP parlance) and generate code that performs much of the work within that component. This greatly simplifies the implementation later on.

Preparation

In order to prepare to specify a design we need to do some folder setup. First, make sure to have followed the class prerequisites including starting the docker container and generating the F´ build cache.

Next, move the existing IMU folder out of the way as we will be replacing it with our own design. We'll replace it with a basic folder with and initial `CMakeList.txt` file inside.

Inside Docker

```
cd ~/fprime-system-reference/SystemReference/Gnc
mv Imu Imu.reference
mkdir Imu
```

Finally, add the following `CMakeLists.txt` to that new folder. The new file must be named exactly `CMakeLists.txt` and exist in the new Imu folder.

```
set(SOURCE_FILES
    "${CMAKE_CURRENT_LIST_DIR}/Imu.fpp"
)
register_fprime_module()
```

We are now setup to generate requirements for our component, codify them into a design, and generate a base implementation. **Note:** the above directory will not build until after we add the FPP file as noted in the design section.

Requirements

Good requirements capture both the behavior of a component and its interaction with other components within the system. Requirements should also capture necessary commands, events, telemetry channels, and parameters for the component. Additionally, failure and off-nominal behaviors should be included in the requirements.

For this exercise, we can follow the GNC sensor integration guide "Step 1: Define Component Requirements" to determine the necessary requirements for the IMU. This section can be found here.

In order to check the requirements we defined, let's compare with the requirements published in the IMU's SDD found here. Note here, we have defined a command, several events, telemetry channels, and defined error conditions.

Design

In F´ the design of the component needs to be formally captured in a model. To do this, we write an FPP file. The design should fall out from the requirements previously captured. Generally, if there are aspects of the design that do not map to requirements, this is an indication that the requirements may be incomplete and should be revisited.

For this exercise, we will follow the design section of the GNC sensor integration guide. We need to make sure to capture the commands, events, telemetry, parameters, and ports of the IMU in our FPP model.

Go ahead and construct an FPP file in your `Imu` directory called `Imu.fpp`. Below is a template you can use to get started. Notice we chose a module of `Gnc` which will result in the Imu being part of the `Gnc` namespace.

```
module Gnc {
    @ The power state enumeration
    enum PowerState {OFF, ON}

    @ Component for receiving IMU data via poll method
    passive component Imu {
```

```
# ----------------------------------------------------------------------
# General ports
# ----------------------------------------------------------------------

@ Port to send telemetry to ground
guarded input port Run: Svc.Sched
...

# ----------------------------------------------------------------------
# Special ports
# ----------------------------------------------------------------------

@ Command receive
command recv port cmdIn

@ Command registration
command reg port cmdRegOut

@ Command response
command resp port cmdResponseOut
...

# ----------------------------------------------------------------------
# Commands
# ----------------------------------------------------------------------

@ Command to turn on the device
guarded command PowerSwitch(
    powerState: PowerState
) \
opcode 0x01


# ----------------------------------------------------------------------
# Events
# ----------------------------------------------------------------------

@ Event where error occurred when requesting telemetry
event TelemetryError(
    status: Drv.I2cStatus @< the status value returned
) \
severity warning high \
format "Telemetry request failed with status {}" \
...

# ----------------------------------------------------------------------
# Telemetry
# ----------------------------------------------------------------------
```

```
        @ X, Y, Z acceleration from accelerometer
        telemetry accelerometer: Vector id 0 update always format "{} g"
        ...
    }
}
```

Note: your design may differ slightly from the example above. Fill in the `...` sections with other parts of your design. The system reference IMU model can be used as a reference to compare our design against.

Generating Implementation Templates

The key advantage of using FPP as a design language is we can now generate the basic implementation of our component. Run the following commands, and you'll see what we mean:

```
cd fprime-system-reference/SystemReference/Gnc/Imu
fprime-util impl
mv ImuComponentImpl.cpp-template Imu.cpp
mv ImyComponentImpl.hpp-template Imu.hpp
```

These template files contain fill-in functions that allow us to implement the logic for the IMU without needing to focus on all the boilerplate code needed to make the `Imu` work with the rest of the system.

Additional Resources

- FPP User Guide
- F´ User Guide
- MPU-6050 Datasheet
- MPU-6050 Register Map

Next Steps

- Component implementation