IMU Exercise: Topology Integration and Testing

The next step in developing our component is to add our component into the system's topology, building it for the RPI, and running the final code.

Topology Integration

In order to see how a component is integrated into topology, we can look through two files in the `Top` folder. These files are `Top/topology.fpp` for the connections and `Top/instances.fpp` used for instantiating components. **Note:** these connections are already in-place as we are working in the system reference. You may delete and recreate them or survey what is present.

In the topology file (`Top/topology.fpp`) we add two instances `imu` and `imuI2cBus` . The first is our `Imu` component and the second is the `imuI2cBus` , which is an instantiation of our `LinuxI2cDriver` .

```
module SystemReference {
  ...

  topology SystemReference {
    ...
    instance imu
    instance imuI2cBus
    ...

    connections I2c {
        imu.read -> imuI2cBus.read
        imu.write -> imuI2cBus.write
    }
    ...
  }
}
```

These instantiations are found in `Top/instances.fpp` where we control the setup of the components. The relevant sections are below. First, we instantiate the IMU component, with a custom configuration phase implementation where we call setup for the component. Next, we instantiate the LinuxI2cDriver with a custom configuration phase to open the RPI device path for I2C.

```
module SystemReference {
  ...
  instance imu: Gnc.Imu base id 0x4C00 {
    phase Fpp.ToCpp.Phases.configComponents """
     imu.setup(Gnc::Imu::I2C_DEV0_ADDR);
     """
  }

  instance imuI2cBus: Drv.LinuxI2cDriver  base id 0x4D00 {
    phase Fpp.ToCpp.Phases.configComponents """
    if (!imuI2cBus.open("/dev/i2c-1")) {
         Fw::Logger::logMsg("[ERROR] Failed to open I2C device\\n");
    }
    """
  }
}
```

Build and Test

Next we can test the system. First we need to perform a complete cross compilation build. Then we upload the binary to our Raspberry PI and run it.

Step 1: Cross-Compile

In this step we will set up a build cache specifically for building for ARM hardware. This is done using generate and passing in the name of the desired toolchain file. Here we pass in `aarch64-linux` . Then we build passing in the same named toolchain.

> Students working with the Raspberry PI 3 may use `arm-hf-linux` in place of `aarch64-linux`

> Mac users must pair with a Linux user or run inside a docker shell for these steps and must then use `/project` instead of `~/fprime-system-reference` in the following instruction.

```
cd ~/fprime-system-reference/SystemReference
```

```
fprime-util generate aarch64-linux
fprime-util build aarch64-linux
```

Step 2: Upload the Built Binary

The next step is to upload the binary file onto the Raspberry PI. Output products are placed in the `build-artifacts/<platform>` directory. In our case the binary is in `build-artifacts/aarch64-linux/bin` . We can upload it using ssh.

```
scp build-artifacts/aarch64-linux/bin/SystemReference odroid@<hostname>:System
```

Note: your team should have been provided the hostname and password for your hardware.

Step 3: Running It!

To run it we need to do two things: launch the GDS and launch the binary on the PI. First, let us launch the GDS. This is done with the `fprime-gds` command using the `-n` flag that prevents the GDS from also running the binary, and we pass in a dictionary, which was automatically built in the build step.

> Macintosh users can launch the GDS outside of docker. Only the cross-compilation must use it.

Launching the GDS

```
cd ~/fprime-system-reference/SystemReference
fprime-gds -n --dictionary build-artifacts/aarch64-linux/dict/SystemReferenceT
```

Next we run the binary and tell it to connect back to the running GDS. This is done from within the PI.

Running the Binary on Hardware

```
ssh odroid@<hostname>
./SystemReference -a <ip of laptop> -p 50000
```

Note: you'll need to determine the IP address of the laptop on the network. **Note:** you may need to open ports on your laptop, if you are running a firewall. Make sure to open port `50000` .

Additional Resources

- F´ User Guide

Next Steps

- Unit Testing