TRENTOS University Course (@ TUM) - SS2022 - Team Projects

Selected project scope

Teams 1, 2 & 3 - Drone Simulator (2-3 students per team)

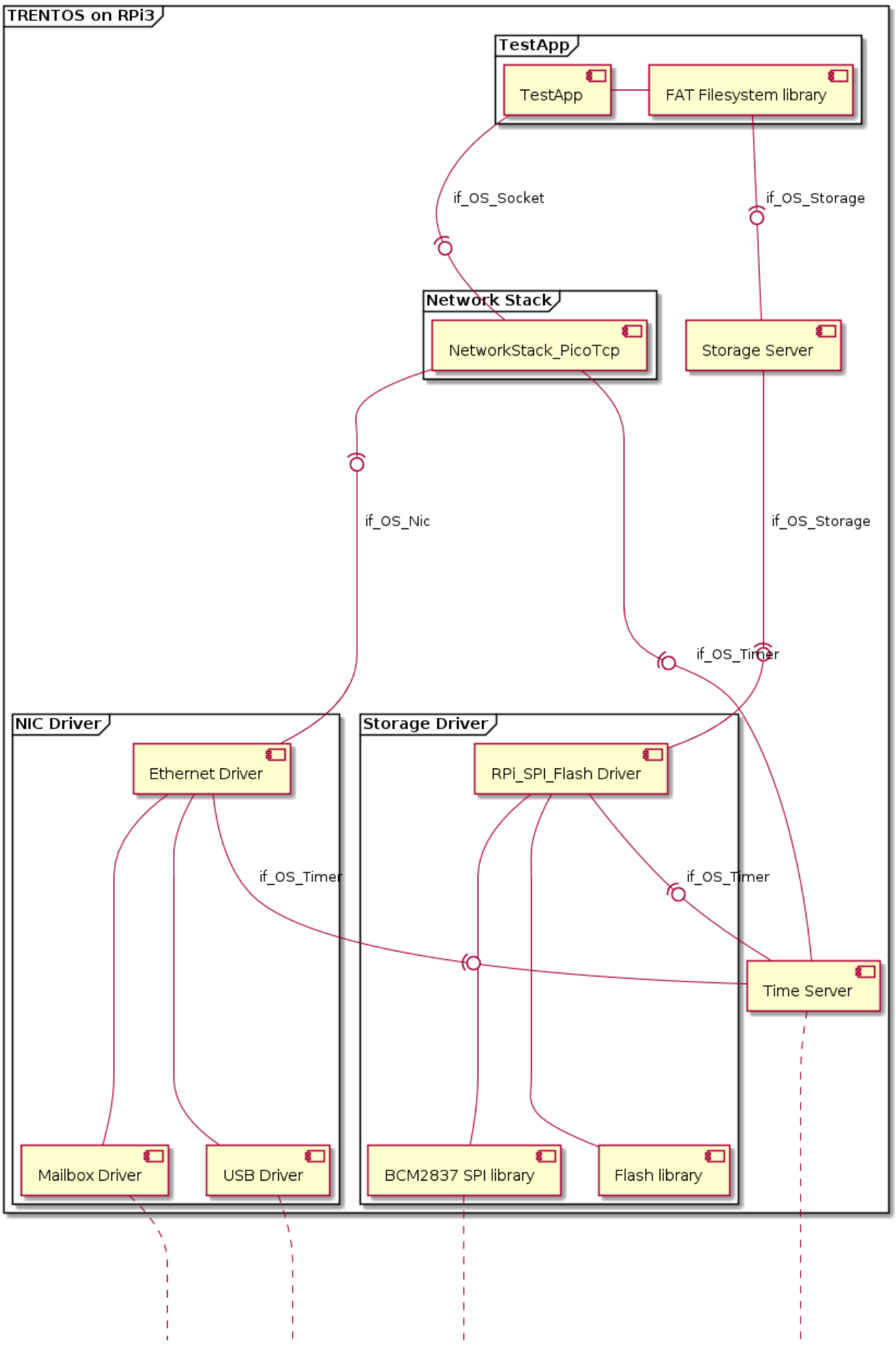
- Project description
  - A main application running on a TRENTOS instance on the RPi 3 B+ shall
    - control a drone in a drone simulator that runs on a Linux PC and
    - search for and detect the highest location to land the drone at
  - Relevant sensor and actuator data is exchanged between TRENTOS and the PC-based drone simulator
    - via Ethernet and
    - by using the API provided by the simulator
- Individual tasks
  - Connection of a drone simulator (AirSim) environment for controlling a virtual drone via TRENTOS that
    - analyzes its environment with the help of a lidar sensor
    - evaluates the lidar data to detect the highest location around it
    - calculates a flight path to this location
    - steers the drone to the planned location and lands it there
  - Depending on the team and the provided hardware, either focus on
    - Implementation of a driver for a SPI-based LCD breakout board in order to visualize the flight direction
      - a Pimoroni SPI LCD breakout board is used
      - an existing driver/library can be ported exemplarily to TRENTOS
    - Implementation of a driver for a I2C-based gyroscope/accelerometer breakout board in order to steer the drone with the help of movements
      - an I2C-based gyroscope/accelerometer breakout board is used
      - an existing driver/library can be ported exemplarily to TRENTOS

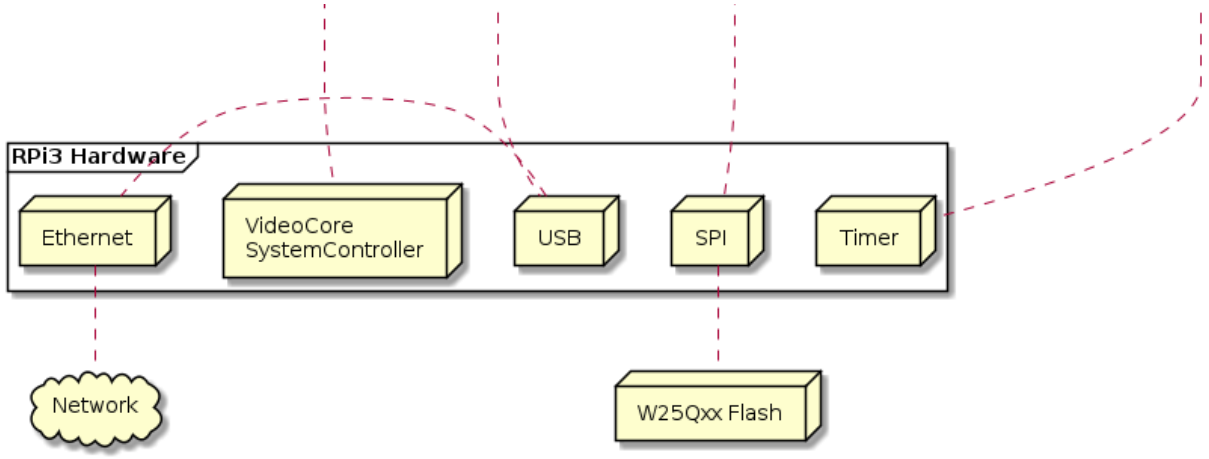
Teams 4 & 5 - TPM (2-3 students per team)

- Project description
  - A main application running on a TRENTOS instance on the RPi 3 B+ shall utilize a hardware-based TPM device to
    - provide access to a TPM-based key store
    - apply hardware-accelerated crypto functionality (in contrast to the existing software-based TRENTOS Crypto API)
- Individual tasks
  - Connection of a SPI-based TPM device to TRENTOS in order to
    - make use of the TPM's random number generator (RNG)
    - access the key store provided by the TPM to be able to store and load keys in a secure way
    - select and utilize a software-based cryptographic algorithm provided by the TRENTOS Crypto API to be able to encrypt/decrypt the content of a file to be stored with the help of the TRENTOS file system support (using the TRENTOS RamDisk component)
    - select and utilize a hardware-accelerated cryptographic function as a replacement to the software-based cryptographic algorithm
    - conduct a performance comparison (here: timing measurement) between software-based and hardware-accelerated approach
  - Implementation of a driver for a SPI-based TPM breakout board in order to encrypt/decrypt file system content
    - a LetsTrust TPM breakout board is used
    - an existing driver/library can be ported exemplarily to TRENTOS

Base Case

Throughout the homework assignments, we have learned all the basics regarding dealing with storage and networking on top of TRENTOS. Let's have a look at the final setup description from Homework Task 5 again:


TRENTOS on RPi3
TestApp
TestApp
FAT Filesystem library
if_OS_Socket
if_OS_Storage
Network Stack
NetworkStack_PicoTcp
Storage Server
if_OS_Nic
if_OS_Storage
if_OS_Timer
NIC Driver
Ethernet Driver
Storage Driver
RPi_SPI_Flash Driver
if_OS_Timer
if_OS_Timer
Time Server
Mailbox Driver
USB Driver
BCM2837 SPI library
Flash library



Initial System Architecture

This system will be our basic setup to start with, regardless of which specific team project task is assigned to you. As already seen before, we can split our initial system into three different logical parts:

- the actual application, which contains our own program logic
- the networking part, consisting of a network driver and the network stack
- the storage part, consisting of a storage driver and the storage server

For the project tasks, we now have to concentrate on three main aspects:

- we want to use the initial setup and extend the networking part (as client) in order to communicate with an external simulator (which is acting as a server)
- we want to use the initial setup and extend the storage part by both software-based and hardware-accelerated encryption/decryption functionality
- depending on the provided hardware, we either want to
    - add a new driver for a SPI-based LCD breakout board (based on the Sitronix ST7735S)
    - add a new driver for an I2C-based gyroscope/accelerometer breakout board (based on the InvenSense MPU-6000)
    - add a new driver for a SPI-based TPM breakout board (based on the Infineon OPTIGA SLB9670 TPM 2.0)

As implementing a new driver from scratch would be too challenging by now, we will concentrate on re-using existing open source drivers (and libraries) and porting them onto TRENTOS. More information will be provided in the sections below.

Project Tasks

For each of the project tasks described above more details regarding the implementation will be provided in the following sections.

Connection of a Simulator

For the connection of the simulator environment to our system, the initial setup provided by the figure above can be re-used. Hereby, only the application (named "TestApp" in the figure above) has to be modified in order to reflect the control logic that is required for communicating with the respective simulator and its virtual sensors and actuators.

Drone Simulator

As already described in the project scope at the beginning, a main application (**`TestApp`**) running on a TRENTOS instance on the RPi 3 B+ shall control a drone in a drone simulator, which is able to fly autonomously to the highest point in its environment in order to land there. Relevant sensor and actuator data must therefore be exchanged between the TRENTOS application and the PC-based drone simulator. The existing TRENTOS system setup shall be re-used to provide Ethernet based communication for interacting with the API provided by the simulator. In our project setup we use the open source robotic simulator AirSim.

What to do

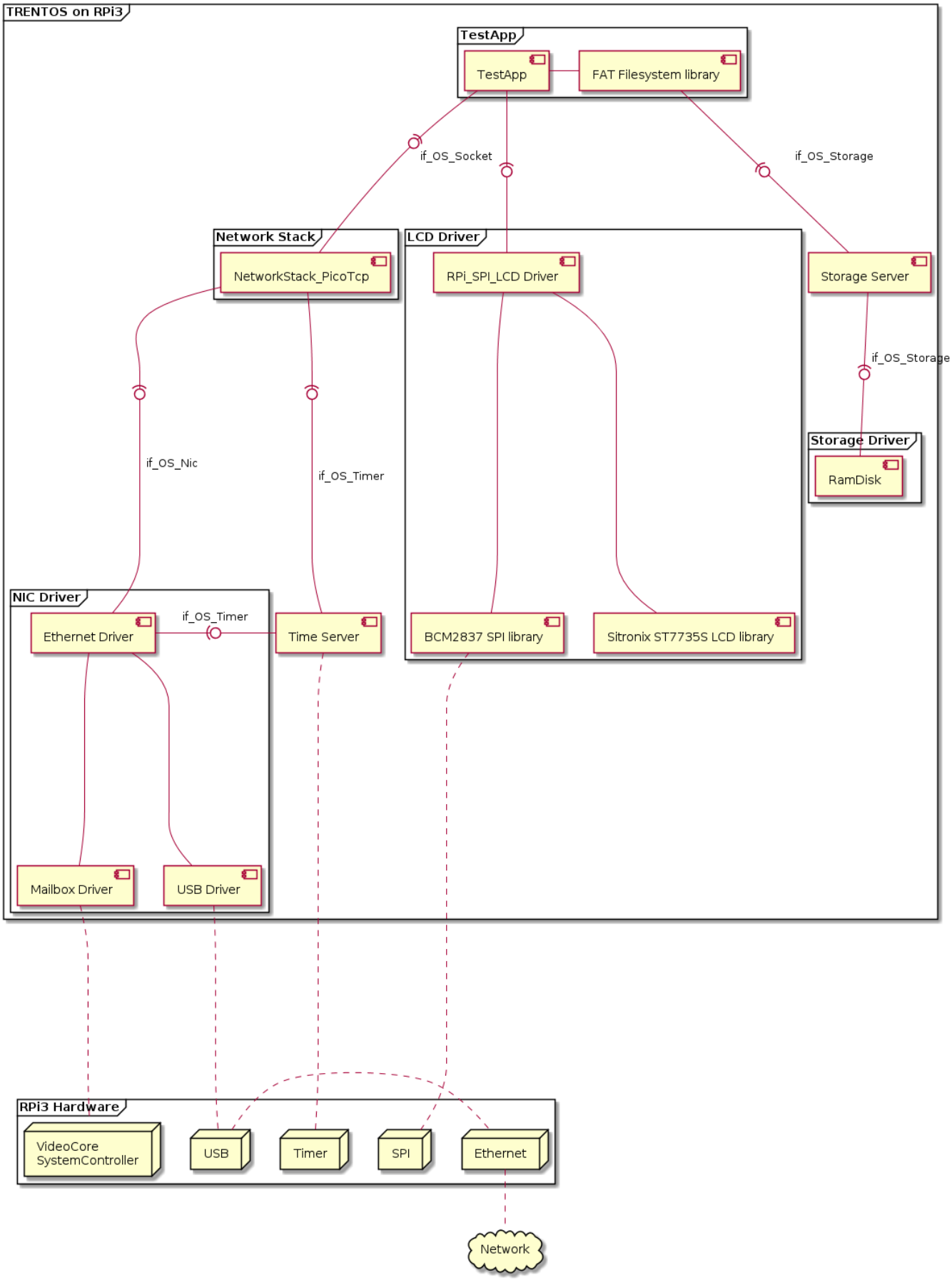
For a successful connection of the simulator to our system, the following steps shall provide a reference:

1. Install AirSim (+ Unreal Engine) on a Linux PC (Ubuntu 18.04 or higher recommended) according to the information provided by the manufacturer. Beware: the utilization of a docker image is strongly recommended. Further information can be found in the AirSim manual, section "Docker on Linux".
2. Select a suitable Unreal environment (see official guide here). In our setup we will start with the simple and lightweight environment Blocks.
3. Read the AirSim manual carefully in order to understand how to
    a. create basic simulation environments
    b. get familiar with the concept of sensors and actuators within the simulator
    c. control the simulator and its objects from the inside
    d. interact with the simulation from the outside via socket communication
    e. start/stop the simulation

4. As we can see, AirSim offers a Python API for communicating with the simulator. Familiarize with connecting to AirSim from outside via the provided API
    a. work through available tutorials
    b. create a small example using the AirSim Python API and the Blocks environment to control a drone in the simulator from a Python script and provide a small scenario for
        - take off
        - flight
        - landing
    c. identify the required sensor data for being able to calculate the small drone scenario (here: using a lidar sensor)
5. Read the AirSim manual for further information regarding
    a. the server side: implemented and offered by AirSim itself
    b. the intermediate side: a Python client application running on the Linux PC that translates from the AirSim Python API to TRENTOS by offering a socket communication; it could optionally be extended to support MQTT on top
    c. the client side: our own application (**`TestApp`**) that shall run on TRENTOS and communicate to the intermediate client on the Linux PC via socket communication (and optionally via MQTT on top)
6. In order to allow for the functionality presented above, we now have to combine the simulator and our **`TestApp`**. We therefore have to
    a. enable the server side within the simulator
    b. get familiar with Python socket programming (e.g. using a tutorial like https://realpython.com/python-sockets/)
    c. create a Python client that acts as intermediate and that
        i. can use the AirSim Python API to connect to the AirSim server, e.g. for reading sensor data or controlling the drone
        ii. allows for socket communication (to the TRENTOS instance)
        iii. provides the respective transition between AirSim Python API and the socket communication
    d. within our **`TestApp`**
        i. use the socket communication (via **`if_OS_Socket`**) via the **`NetworkStack_PicoTcp`** component (see TRENTOS Handbook, chapter "TRENTOS Socket API") to receive the required sensor data (the lidar data) from the simulator via the intermediate Python client
        ii. optionally utilize MQTT on top of the socket communication
        iii. provide the necessary calculations for the control of the drone (analyze the lidar data, calculate the flight path, navigate the drone to its target platform and land there)
        iv. send the corresponding actuator commands from the **`TestApp`** to the intermediate client on the Linux PC, which will finally forward them to the simulator
    e. Depending on the team and the provided hardware (see respective driver sections below) either
        i. visualize the flight direction via the SPI-based LCD breaktout board (e.g. in form of drawn arrows) or
        ii. accept control input from the I2C-based gyroscope/accelerometer breakout board, e.g. by moving/turning it in different directions, in order to influence the flight speed of the drone
7. Stop the demo after the drone has successfully landed on the highest point within the environment. When the demo is finally stopped, read the amount of objects sorted by color from the storage and print them on a terminal.

Driver for a SPI-based LCD breakout board

The respective initial setup (demonstrated in section "Base Case") we want to use for porting the SPI-based LCD breakout board has a small limitation. As TRENTOS currently only supports one SPI-based peripheral at the same time, we have to switch from the SPI NOR Flash storage driver to the RamDisk again. The figure below therefore depicts the respective target system architecture:



RPi_SPI_LCD Driver - Architecture

What to do

For a successful driver development and integration into our system, the following steps shall provide a reference:

1. As we can only support one SPI-based peripheral for now, replace the SPI NOR Flash driver component with the **`RamDisk`** component (TRENTOS Handbook, chapter "TRENTOS RamDisk") in order to start with the initial setup.
2. Provide the correct wiring of the SPI-based LCD breakout board to the RPi 3 B+. Please use the wiring of the SPI NOR Flash described in the TRENTOS Handbook, chapter "TRENTOS RPi_SPI_Flash" as a reference.
3. Nevertheless, the driver internals still have to be adapted to the new Sitronix ST7735S based hardware device. As we want to provide support for a SPI-based peripheral, please have a look at the **`RPi_SPI_Flash`** component (see TRENTOS Handbook, chapter "TRENTOS RPi_SPI_Flash"). It can be considered as a starting point regarding the driver implementation for SPI-based peripherals in general, as it already provides the required abstractions and support for the respective low-level mechanics:
    a. the GPIO handling (see ***`bcm2837_gpio.h`*** and ***`bcm2837_gpio.c`***)
    b. the SPI protocol (see ***`bcm2837_spi.h`*** and ***`bcm2837_spi.c`***)
4. On top of the GPIO and SPI abstractions, we then have to provide the actual peripheral-specific functionality. Within the **`RPi_SPI_Flash`** component, this is wrapped into a peripheral-specific library, e.g. here for using a flash storage on top of the SPI protocol (see ***`spiflash.h`*** and ***`spiflash.c`***). We can do the same for our SPI-based LCD peripheral (see section "Potential resources" below for more information).
5. Finally, we have to create a TRENTOS specific wrapper on top of this which represents the actual "driver" component within the OS (for the **`RPi_SPI_Flash`** component: ***`RPi_SPI_Flash.c`*** and ***`RPi_SPI_Flash.camkes`***).
6. All other components presented in the figure above can be re-used without any modifications.
7. We can test our driver implementation with the help of our target system architecture and the final test application of Homework Task 5.

Potential resources

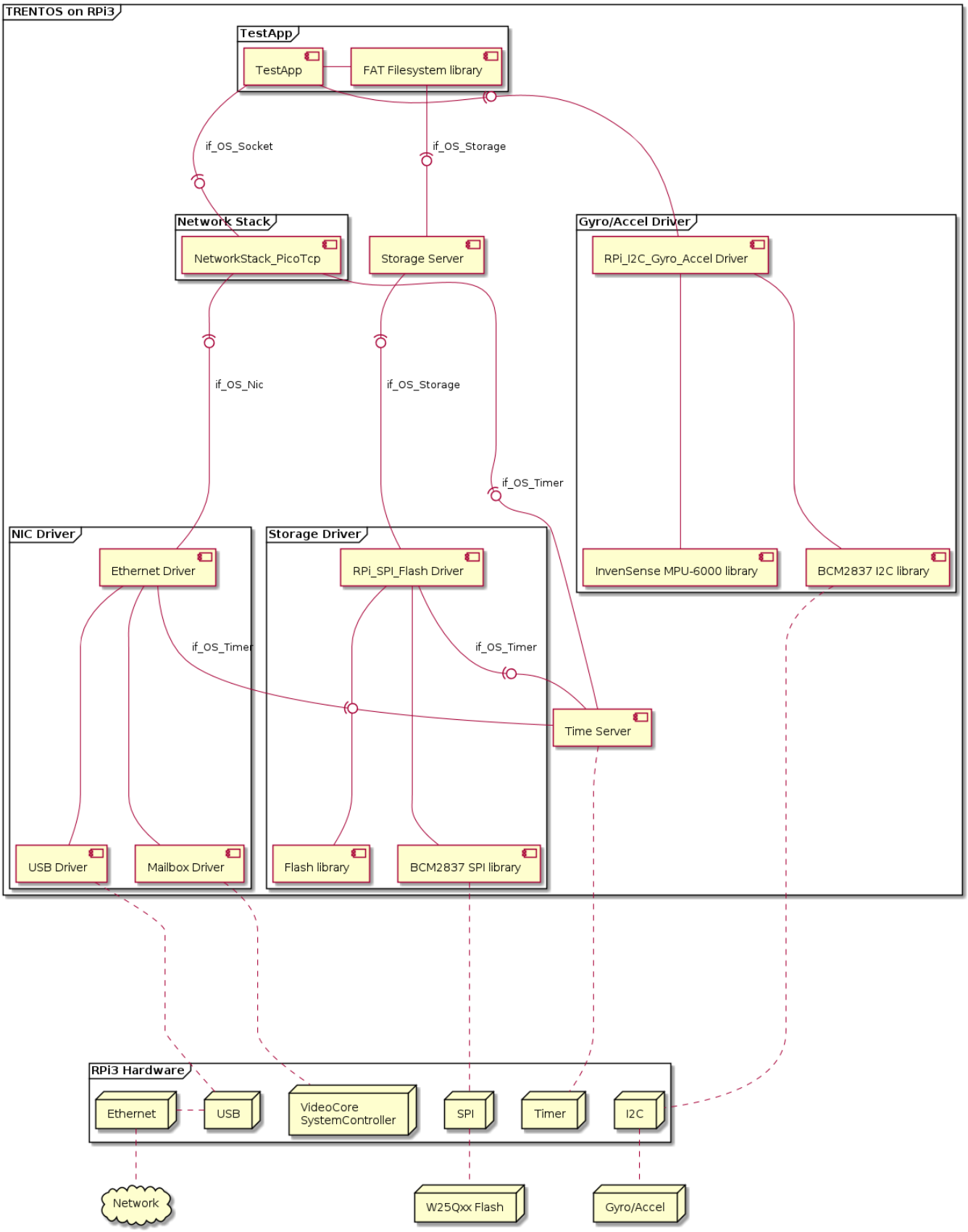
The SPI-based LCD breakout board we will utilize for the project task is based on the Sitronix ST7735S. Please have a look at the following resources, which provide helpful information and source code examples for driver/library development based on this specific chipset. These resources can be used as a starting point for implementing (or rather porting) your own driver/library to TRENTOS.

Note: don't remove any license related information (e.g. GPL statements) and document from which sources you have taken code or code snippets for your own driver!

- https://github.com/bersch/ST7735S
- https://github.com/Matiasus/ST7735

Driver for an I2C-based gyroscope/accelerometer breakout board

The respective initial setup we want to use for porting is demonstrated in section "Base Case". The figure below depicts the respective target system architecture.



RPi_I2C_Gyro_Accel Driver - Architecture

What to do

For a successful driver development and integration into our system, the following steps shall provide a reference:

1. Provide the correct wiring of the I2C-based gyroscope/accelerometer breakout board to the RPi 3 B+. Please use the wiring described here as a reference.
2. As there is currently no I2C driver support for the RPi 3 B+ integrated into the SDK itself, please have a look at the provided ***i2c_driver_demo.zip*** file, which can be used as a starting point for development. The demo consists of a general I2C driver component (**i2c**) and library support for two example peripherals - a temperature/pressure/humidity sensor (**bmp280**) and a OLED device (**ssd1306**). The I2C driver shall be reused and from the two provided examples an idea for our new I2C-based gyroscope/accelerometer device shall be derived.
3. We therefore have to create a new component that provides support for our InvenSense MPU-6000 based hardware device. The generic I2C driver already provides the required abstractions and support for the respective low-level mechanics:
    a. the GPIO handling (see ***bcm2837_gpio.h*** and ***bcm2837_gpio.c***)
    b. the I2C protocol (see ***bcm2837_i2c.h*** and ***bcm2837_i2c.c***)
    c. the platform dependencies (see subfolder ***/include/plat/rpi3***)
    d. the I2C driver itself (***i2c.c*** and ***i2c.camkes***)
4. On top of the driver, we then have to provide the actual peripheral-specific functionality. Within the I2C demo components, this is wrapped into peripheral-specific libraries, e.g. here for using either the **bmp280** or the **ssd1306** on top of the I2C protocol (see respective subfolders within the demo's subfolder ***/components***). We shall now do the same for our I2C-based gyroscope/accelerometer peripheral (see section "Potential resources" below for more information).
5. We can test our driver implementation with the help of the provided demo and afterwards based on our target system architecture and the final test application of Homework Task 5.

Potential resources

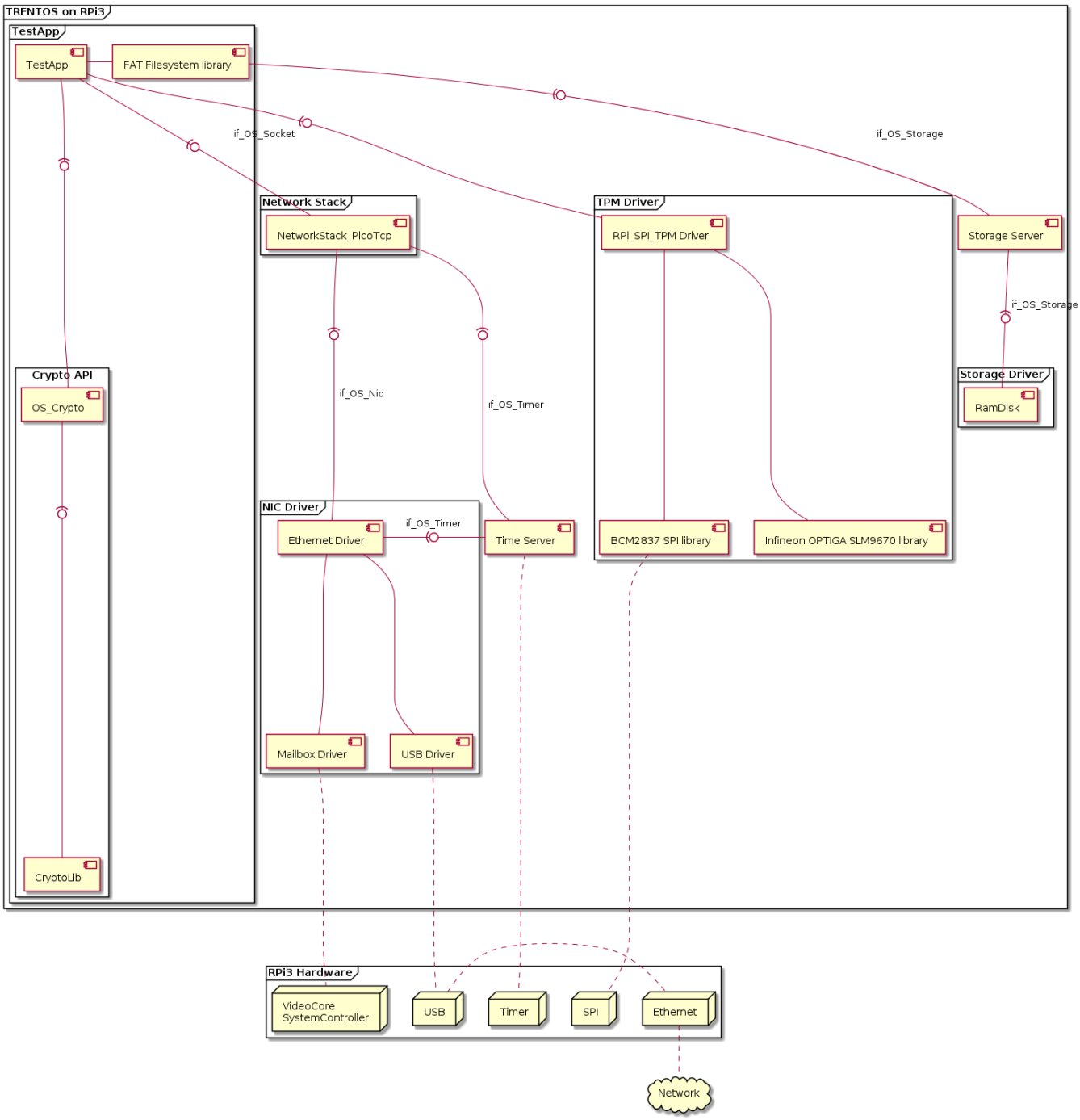
The I2C-based gyroscope/accelerometer breakout board we will utilize for the project task is based on the InvenSense MPU-6000. Please have a look at the following resources, which provide helpful information and source code examples for driver/library development based on this specific chipset. These resources can be used as a starting point for implementing (or rather porting) your own driver/library to TRENTOS.

Note: don't remove any license related information (e.g. GPL statements) and document from which sources you have taken code or code snippets for your own driver!

- https://github.com/jrowberg/i2cdevlib/blob/master/PIC18/MPU6050/MPU6050.c
- https://github.com/ElectronicCats/mpu6050
- https://github.com/jarzebski/Arduino-MPU6050
- https://github.com/yuvadm/tiva-c/blob/master/sensorlib/mpu6050.c
- https://github.com/MarkAYoder/BeagleBoard-exercises/blob/master/sensors/imu/mpu6050.c
- https://github.com/nabto/unabto/blob/master/3rdparty/nuvoton/MPU6050.c
- https://github.com/wennycooper/mpu6050_in_c

TPM

The respective initial setup (demonstrated in section "Base Case") we want to use for porting the SPI-based TPM breakout board has a small limitation. As TRENTOS currently only supports one SPI-based peripheral at the same time, we have to switch from the SPI NOR Flash storage driver to the RamDisk again. The figure below therefore depicts the respective target system architecture:



RPi_SPI_TPM Driver - Architecture

As already described in the project scope at the beginning, a main application (**`TestApp`**) running on a TRENTOS instance on the RPi 3 B+ shall utilize a hardware-based TPM device to provide access to a TPM-based key store and to apply hardware-accelerated crypto functionality (in contrast to the existing software-based TRENTOS Crypto API available in the SDK). The existing TRENTOS system setup shall be re-used to provide Ethernet based communication for interacting with the PC and for receiving input data based on our webserver example from homework 5. The received data shall then be stored in an encrypted fashion into a file that resides on the connected RamDisk backend.

What to do

For a successful driver development and integration of the LetsTrust TPM breakout board into our system, the following steps shall provide a reference:

1. As we can only support one SPI-based peripheral for now, replace the SPI NOR Flash driver component with the **`RamDisk`** component (TRENTOS Handbook, chapter "TRENTOS RamDisk") in order to start with the initial setup.
2. Provide the correct wiring of the SPI-based TPM breakout board to the RPi 3 B+. Please use the wiring of the SPI NOR Flash described in the TRENTOS Handbook, chapter "TRENTOS RPi_SPI_Flash" as a reference.

3. Nevertheless, the driver internals still have to be adapted to the new Infineon OPTIGA SLB9670 TPM 2.0 based hardware device. As we want to provide support for a SPI-based peripheral, please have a look at the **`RPi_SPI_Flash`** component (see TRENTOS Handbook, chapter "TRENTOS RPi_SPI_Flash"). It can be considered as a starting point regarding the driver implementation for SPI-based peripherals in general, as it already provides the required abstractions and support for the respective low-level mechanics:
    a. the GPIO handling (see ***`bcm2837_gpio.h`*** and ***`bcm2837_gpio.c`***)
    b. the SPI protocol (see ***`bcm2837_spi.h`*** and ***`bcm2837_spi.c`***)
4. On top of the GPIO and SPI abstractions, we then have to provide the actual peripheral-specific functionality. Within the **`RPi_SPI_Flash`** component, this is wrapped into a peripheral-specific library, e.g. here for using a flash storage on top of the SPI protocol (see ***`spiflash.h`*** and ***`spiflash.c`***). We can do the same for our SPI-based TPM peripheral (see section "Potential resources" below for more information). It is recommended to have a look at the open-source TCG TPM2 Software Stack and the open-source wolfTPM library.
5. Finally, we have to create a TRENTOS specific wrapper on top of this which represents the actual "driver" component within the OS (for the **`RPi_SPI_Flash`** component: ***`RPi_SPI_Flash.c`*** and ***`RPi_SPI_Flash.camkes`***).
6. Examine both the mbedTLS library (which we use within the TRENTOS Crypto API) and the Infineon OPTIGA SLB9670 TPM 2.0 to select a cryptographic algorithm that is supported by both variants, e.g. the RSA1024.
7. We now have to adapt our **`TestApp`**, in order to
    a. support the TRENTOS Crypto API in **`OS_Crypto_MODE_LIBRARY`** mode (see TRENTOS Handbook, chapter "TRENTOS Crypto API" as a reference), to allow for software-based crypto support
    b. provide a connection to our new TPM driver component to allow for hardware-accelerated crypto support
8. The **`TestApp`** shall then
    a. make use of the TPM's random number generator (RNG)
    b. create required encryption keys and store them in the TPM's key store
    c. receive data (with a reasonable size) from the PC (by reusing the webserver example)
    d. encrypt the received data by using the previously created keys and with the help of
        i. the software-based crypto support
        ii. the hardware-accelerated crypto support
    e. measure the time required for each encryption approach
    f. store the encrypted data into two separate files on the RamDisk using the existing TRENTOS FileSystem support
    g. load the data from the files and decrypt it in oder to verify everything went well
9. All further components presented in the figure above can be re-used without any modifications.
10. We can test our driver implementation with the help of our target system architecture and the final test application of Homework Task 5.

Potential resources

The SPI-based TPM breakout board we will utilize for the project task is based on the Infineon OPTIGA SLB9670 TPM 2.0. Please have a look at the following resources, which provide helpful information and source code examples for driver/library development based on this specific chipset. These resources can be used as a starting point for implementing (or rather porting) your own driver/library to TRENTOS.

Note: don't remove any license related information (e.g. GPL statements) and document from which sources you have taken code or code snippets for your own driver!

- https://letstrust.de/
- https://www.infineon.com/cms/de/product/security-smart-card-solutions/optiga-embedded-security-solutions/optiga-tpm/slm-9670/
- https://www.infineon.com/cms/de/product/security-smart-card-solutions/optiga-embedded-security-solutions/optiga-tpm/slb-9670vq2.0/#!documents
- https://www.infineon.com/dgdl/Infineon-SLM%209670-DataSheet-v01_00-EN.pdf?fileId=5546d46269e1c019016a21819cc80c93
- https://www.infineon.com/dgdl/Infineon-product-brief-optiga-tpm9670-PB-v01_00-EN.pdf?fileId=5546d462696dbf1201697c89f4bb4789
- https://www.infineon.com/dgdl/Infineon-App-Note-SLx9670-TPM2.0_Embedded_RPi_DI_SLx-ApplicationNotes-v01_03-EN.pdf?fileId=5546d46267c74c9a01684b96e69f5d7b
- https://www.infineon.com/cms/de/product/evaluation-boards/iridium-slm-9670-tpm2.0/
- https://github.com/tpm2-software/tpm2-tss
- https://github.com/wolfssl/wolfTPM
- https://ubs_csse.gitlab.io/secu_os/tutorials/tpm_rpi.html

Final Integration

In order to demonstrate a fully working showcase, the individual project parts of each team have to be integrated in order to form a complete setup. As already mentioned, TRENTOS currently only supports one SPI-based peripheral at the same time. Therefore, the final integration setup differs per team depending on the provided SPI-based hardware.

Drone Simulator Setup

Team 1 & 2 - SPI-based LCD

The setup consists of the following parts:

- the drone simulator
- the target system architecture provided in the section "Driver for a SPI-based LCD breakout board", consisting of the
    - **`NIC_RPi`** component
    - RamDisk component
    - SPI-based LCD driver
    - **`TestApp`** for controlling the drone simulator

Team 3 - I2C-based gyroscope/accelerometer

The setup consists of the following parts:

- the drone simulator
- the target system architecture provided in the section "Driver for an I2C-based gyroscope/accelerometer breakout board", consisting of the
  - **`NIC_RPi`** component
  - **`RPi_SPI_Flash`** component
  - I2C-based gyroscope/accelerometer driver
  - **`TestApp`** for controlling the drone simulator

TPM Setup

Team 4 & 5 - SPI-based TPM

The setup consists of the following parts:

- the target system architecture provided in the section "TPM", consisting of the
  - **`NIC_RPi`** component
  - RamDisk component
  - SPI-based TPM driver
  - **`TestApp`** for controlling the TPM