Ben Gurion University

NegevSat

Platform for Satellite Software

Final Document

Hod Amran

Yaniv Radomsky

August 2015

Contents

1. Maintenance Guide

1.1 Communication Package:

- **CommuncationHandlerFactory –** Responsible for creation of CommunicationHandler, using the Factory design pattern. Creation is performed using the createHandler method. For now, the only implementation is UartCommunicationHandler. In order to add another implementation the user should create a class which implements the ICommunicationHandler interface and add a case in the createHandler method.
- **SendReceiveQueue** – A queue which stores the incoming and outgoing data of the system.
  The main methods of this class are:
  - enqueue(string msg) – enqueues a message msg, if the queue is full overrides the oldest message in the queue. The size can be adjusted using the SEND_RECEIVE_QUEUE_SIZE constant.
  - dequeue() – dequeues a message from the queue, if the queue is empty blocks until messages arrive.
- **UartCommunicationHandler** – Implements ICommunicationHandler, used for send and receive data from the communication channel.

1.2 Data Protocol Package:

- **abstract_datatype Package**- this package contain all abstract data type of packet that sent from satellite to the ground station. This packet contain 4 class:
  - Packet – an abstract class that represent a packet structure and methods.
  - PacketFactory – an abstract Factory class that responsible for creation a packet.
  - EnergyPacket – represent an energy packet and extends packet abstract class.
  - StatuesPacket – represent an Statues packet and extends packet abstract class
  - TempsPacket – represent an Temperature packet and extends packet abstract class.
- **ICMDParser** – interfase od command parser , contain method parsePacket for parsing an incoming packet and split it into readable "works".

- **binary_protocol Package** – this package contain all class that is necessary for parsing command in binary parsing, it contains the following class :

  - BinCMDParser – extends ICMDParser, the main class that is responsible for parsing packet in binary code.
  - BinPacketFactory – exstends PacketFactory in binary way,means that this is a binary packet factory for creating a binary packet.
  - BinEnergyPacket – extends EnergyPacket in binary way, means that this is a binary coded, in structure of energy packet.
  - BinStatuesPacket – extends StatuesPacket in binary way, means that this is a binary coded, in structure of Statues packet.
  - BinTempsPacket – extends TempsPacket in binary way, means that this is a binary coded, in structure of Temps packet.

- **Xml_protocol Package** – this package contain all class that is necessary for parsing command in Xml parsing, it contains the following class :
  - Xml CMDParser – extends ICMDParser, the main class that is responsible for parsing packet in Xml code.
  - Xml PacketFactory – extends PacketFactory in binary way, means that this is a Xml packet factory for creating a binary packet.
  - Xml EnergyPacket – extends EnergyPacket in binary way, means that this is a Xml coded, in structure of energy packet.
  - Xml StatuesPacket – extends StatuesPacket in binary way, means that this is a Xml coded, in structure of Statues packet.
  - Xml TempsPacket – extends TempsPacket in binary way, means that this is a Xml coded, in structure of Temps packet.

- **WorkQueue** - A queue which stores the work objects which should be performed by the system.
  - The main methods of this class are:
  - enqueue (WorkDescription work) – enqueues a work, if the queue is full overrides the oldest work in the queue. The size can be adjusted using the QUEUE_SIZE constant.
  - dequeue(bool block) – dequeues a work from the queue, if the block flag is true and the queue is empty - blocks until work arrives, otherwise will return a NULL work without blocking.
  - sortWorks() – sort the works in the queue by their time of execution. When a work's time of execution is set to zero – the system will immediately perform this work.
- **CMDParser** – used in the last version of negevsat in 2014 , can be raised for more details look in 2014 Docs.

- **TLMParser** – used in the last version of negevsat in 2014 , can be raised for more details look in 2014 Docs.
- **`TempPacket`** – used in the last version of negevsat in 2014 , can be raised for more details look in 2014 Docs.
- **ValidatorFactory** - used in the last version of negevsat in 2014 , can be raised for more details look in 2014 Docs.
- **XMLValidator** – used in the last version of negevsat in 2014 , can be raised for more details look in 2014 Docs.

1.3 Hardware Package:

This package is used in the negevsat 2014.

- HWModule – this class represents a HW module, all specific modules (e.g. EnergyModule, PayloadModule) extends this class. In future this class implementation should be replaced with hardware calls (Stub class for now).
- HardwareState – contains all hardware modules in the system, all accesses to the modules are performed via this class. All getter methods with “useI2C” argument will perform hardware call if the argument is ‘true’ else will read from the cache.

1.4 Hardware2 Package:

This package is used in the negevsat 2015.

- **Modules Package**- this package contains all modules of satellite based on PEASSS documents. This packet contain the class:
    - Antenna– represents the antenna module in the satellite.
    - AttitudeControl – an interface that represents the attitude Controller module of the satellite.
    - AttitudeControlStub – implements the AttitudeControl, and is stub for modules that control the attitude of the satellite.
    - ThermalControl - an interface that represents the Thermal Controller module of the satellite.
    - ThermalControlStub – implements the ThermalControl, and is stub for modules that control the attitude of the satellite.

- **Sensors Package**- this package contains all sensors of satellite based on PEASSS documents. This packet contain the class:
    - ISensor – an interface that represents a sensor in satellite.
    - AttitudeSensor – implements ISensor , represents Attitude sensor of satellite.
    - CurrentSensor– implements ISensor , represents Current sensor of satellite.

- TemperatureSensor – implements ISensor , represents Temperature sensor of satellite.
- SunSensorSensor – implements ISensor , represents the sun exposer sensor of satellite.
- DodSensor – implements ISensor , represents the Dod(Depth of Discharge) sensor of satellite.

- HardwareStub – this is the main hardware class that contain all the modules and sensors of the satellite, and manages the transport of call to the component and back to the system.

1.5 Logics Package:

- Global.hpp – contains the global data types which are shared between all tasks, contains the task_table and the time structure for now.
- nevSat.cpp – the main class does the following operation:
  - Configures the current time and date.
  - Performs tests if the bool flag ‘tests’ is set to ‘true’.
  - Creates and starts all tasks.
  - Initializes the task_table with the tasks.
  - Destroys itself.
  - Note: information about the RTEMS defines (in the head of the file) can be found in c_user.pdf manual inside the rtems documentation library.

- NegevSatConstants.hpp – contains all global constants which are used in the system.
- SendReceiveConf.hpp – defines the send queues constants, there are 3 at this moment.

1.5.1 Tasks Package

Contains all Task objects (equivalent to Java Thread class) of the system. The “run” method of each task is called ‘body’. In order to add a new task to the system the user should create a new class which extends rtemsTask and implement the body method (see other tasks implementation).

In addition, the following steps should be taken:

1. Increase the NUMBER_OF_TASKS constant inside the ‘NegevSatConstants.hpp’ file.
2. Change the NUMBER_OF_PASSIVE_TASKS accordingly.
When NUMBER_OF_PASSIVE_TASKS means the number of tasks which should not receive events from the state machine task (e.g. the StateMachineTask).
3. Add a unique task index constant under the ‘//Tasks constants’ in the ‘NegevSatConstants.hpp’ file.

4. In ‘negevSat.cpp’ file create the task, start the task and add the task to the task table (see other tasks in the file).

There are 6 tasks in the system:

- CMD Task – this task constantly sorts the works in the WorkQueue, dequeues the earliest work, checks whether the time of the work has come, if so enqueues the work to the ready work queue. Otherwise returns the work back to the queue.
- LifeCycle Task – this task is responsible for the satellites autonomous functionality, including the following functions:
  - obtain_state() – obtains the state using the RTEMS events, which are sent by the StateMachine Task.
  - attitude_control() – performs these 3 actions:
    - control_unit_samples – create empty packet using the telemetry parser and fill the packet with Sample objects using the Sampler object (measures).
      The constant MAX_SAMPLES stands for the number of samples in each packet.
    - control_algorithmics – not implemented at this moment. In future should perform control algorithm to the measures from the previous function.
    - control_command – writes the result of the algorithms calculation to the – right now deals with safe state only.
  - logics() - changes state according to health level of the system (e.g. when the energy is too low the state will be changed to Safe state).
  - perform_cmd() – executes a work from the ready work queue using the CommandExecutor object.
  - monitoring() – checks the modules condition and sets their status to MALFUNCTION if needed.
  - module_ctrl() – turns on/off modules bychecking the ModuleOperationRequest object – if there is a request to turn a module on and the state of the system allows this operation then turn on the module.

    Note: all code under the // SIMULATOR FIELDS comment is not relevant to the system but for the simulation only (should be removed later).
  - thermal_control() – turns on the thermal control if needed.
- MP Task – Message Parser Task – dequeues a packet from the receive queue. Constructs a packet using the Validator. Validates the packet, if the

packet is valid parses the packet using the CMDParser and enqueues the works to the work queue, otherwise displays an error message and continues.

- Receive Task - this task constantly listening to the communication channel using the CommunicationHandler, once there is data on the channel verifies the bytes and enqueues the message to the ReceiveQueue.
  Note: if there is no data on the channel waits for data (blocking).

- Send Task – first this task obtains the state of the system (checks if the satellite faces the ground station or not). Then obtains the send type (more in section 2.5), if the system faces the ground station sends the packets according to the send type.

- StateMachine Task – this task uses the Macho state machine third party library (see documentation at http://ehiti.de/machine_objects/). This task informs all other tasks about the system state by sending events when the state machine enters a specific state. In addition, this task listens to events from the other tasks and moves between the states accordingly.

1.5 Tests Package:

Tests are based on Minunit testing framework (see documentation at http://www.jera.com/techinfo/jtns/jtn002.html).

Alltests.cpp contains all tests classes, the method all_tests defines which tests will run according to the argument ‘type’ when executing the tests.

The method run_all_tests executes the tests and checks if they passed or failed.

In order to add a test, create a new class which includes “minunit.h”, implement char* runTests() method, add the class to “Alltests.hpp” fields, add a case to all_tests method (see other test as an example).

2. User Guide

2.1 Environment Setup For Windows:

2.1.1 Install GRTools

Detailed instructions can be found here:
http://gaisler.com/doc/gr_lide.pdf

1. Download GRTools from:
   http://www.gaisler.com/index.php/products?task=view&id=272
2. Install the package.
3. Assert that "Path" environment variable was updated correctly - it should contain the following paths:
   - mkprom2
   - tsim
   - grmon
   - rtems
   - sparc-elf
   - bin folder of the installation (default is c:\opt\bin)

If any of the paths are not correct, fix it manually.

2.1.2 Update GRTools Components:

Sometimes the GRTools components are not fully updated for example Tsim is old release and not working any more.
So we need to upgrade them manually:

1. Download latest release of TSIM:
   http://www.gaisler.com/index.php/downloads/simulators
   under "TSIM2 LEON3 Evaluation" Title.
2. Copy tsim-eval folder to c:\opt and replace content

3. Download latest release of RTEMS LEON/ERC32 compiler:
   http://www.gaisler.com/anonftp/rcc/bin/mingw/
4. Copy rtems-4.10-mingw folder to c:\opt and replace content.
   Note:If there is any problem with compiler try older version (we used RCC-1.2.18. RTEMS-4.10, GCC 4.4.6 version).

2.2 Test Environment

2.2.1 Quick hello-world C++ application:

1. Run Eclipse which was installed in the previous tutorial.
2. Create a new C++ project:
   - 2.1 File -> new -> C++ project.
   - 2.2 Choose a name and type it in the "Project name" field.
   - 2.3 Under Project type select "Hello World C++ Project".
   - 2.4 Under Toolchains select SPARC RTEMS.
   - 2.5 Press finish.
3. Setting compiler flags:
   - 3.1 Right click on the project in the project explorer and press properties.
   - 3.2 Under C/C++ Build -> Settings -> SPARC RTEMS C++ Compiler -> Hardware mark the V8 extension.
4. Setting Debug configuration:
   - 4.1 Go to Run -> Debug Configurations -> LEON C/C++ application and add your project there.
   - 4.2 Once your project appears under LEON C/C++ application click on it and go to the Debugger window.
   - 4.3 Add external target by clicking on "New..." near the External target field.
   - 4.4 Make sure the target's name is TSIM2 and Browse to the location of the TSIM simulator - the default location is C:\opt\tsim-eval\tsim\win32\tsim-leon3.exe.
   - 4.5 Press on finish.
5. Debug your project and see the output of TSIM in your console.

2.3 Configuring the project:

1. Clone the project from GitHub using this link:
   https://github.com/yannn12/NegevSat-project2015-new
2. Import the project into eclipse (the project should be RTEMS project as explained in the previous tutorial).
3. TSIM2 Configurations:
   - 3.1 Enter Debug Configurations
   - 3.2 Press the project name under the LEON C/C++ application.
   - 3.3 Press on the Debugger tab
   - 3.4 Under ‘External Target’ press the ‘Edit’ button (if the target was not configured, follow the previous tutorial step 4).
   - 3.5 Inside the ‘Extra options’ field add the following flags:

"-timer32 -fast_uart -uart2 //./com1"
Explanation for these flags and more can be found in the TSIM2 documentation.

4. Now the project can be debugged (only debug works).

2.4 Running the Simulator:

1. Clone the project from GitHub using this link:
   https://github.com/yannn12/NegevSat-Simulator-2015/
1. Download java jdk v8 32 bit.
2. Set an environment variable JAVA_HOME to point on that java.
   For example: C:\Program Files (x86)\Java\jdk1.8.0_25.
3. install RXTX serial port to java
   3.1 Download it :
       http://rxtx.qbang.org/pub/rxtx/rxtx-2.1-7-bins-r2.zip
   3.2 install it to JAVA_HOME/bin by this guide:
       http://rxtx.qbang.org/wiki/index.php/Installation_for_Windows

4. Once the source code is cloned, enter the simulator " NegevSat-Simulator-2015\jar" folder and run one of the batch file as needed.
5. You can edit the .bat file for custom configurations

Note: if you don't have physical serial port on your pc follow the instructions in the next tutorial.

2.5 Communication Guide:

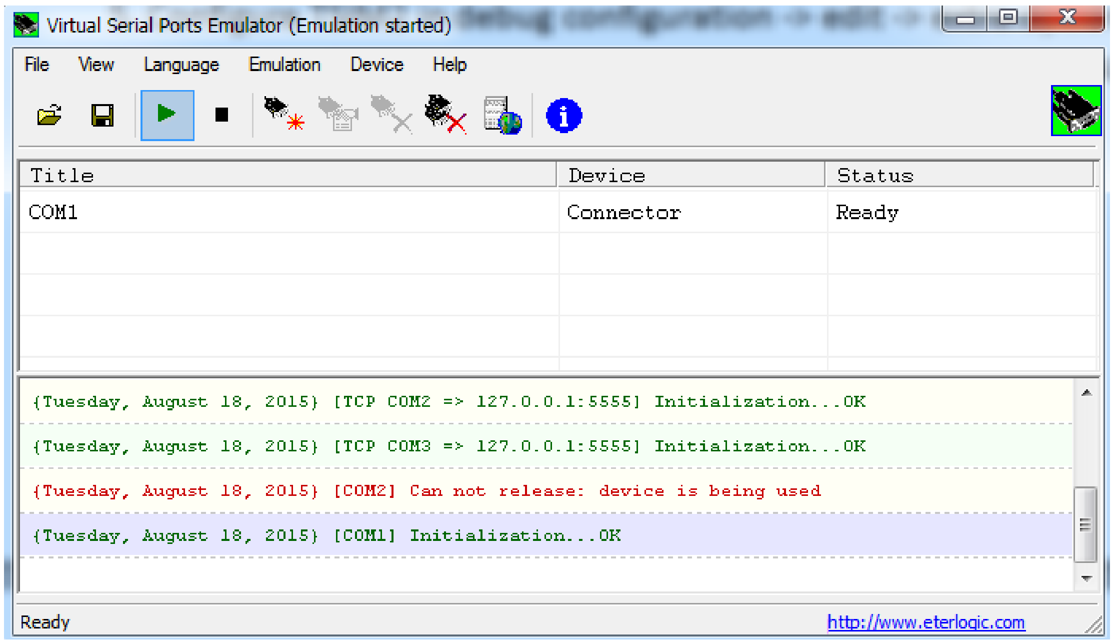
1. Download and install VSPE software (Virtual Serial Ports Emulator).
2. (Optional)Download and install TeraTerm software (For tests).
3. On server (airborne system) configure virtual COM1 port and then TCP server listening on that port (using VSPE).
4. On clients (ground station and simulator) configure virtual COM1 port and then TCP client listening to the server ip and port (usually 5555).
5. Configure TSIM2 in debug configuration -> edit -> extra options with -fast_uart flag (EXTRA IMPORTANT! WITHOUT THIS FLAG THE DATA WILL GET LOST).

2.6 Airborne Software System Configurations:

2.6.1 Airborne Software with Simulator configure(for tests only)

This working mod is used to test the Airborne Software with simulator only without Ground Station connected.

1. Configure VSPE as define below


Virtual Serial Ports Emulator (Emulation started)
File View Language Emulation Device Help
Ready
http://www.eterlogic.com

Title	Device	Status
COM1	Connector	Ready

```
{Tuesday, August 18, 2015} [TCP COM2 => 127.0.0.1:5555] Initialization...OK
{Tuesday, August 18, 2015} [TCP COM3 => 127.0.0.1:5555] Initialization...OK
{Tuesday, August 18, 2015} [COM2] Can not release: device is being used
{Tuesday, August 18, 2015} [COM1] Initialization...OK
```

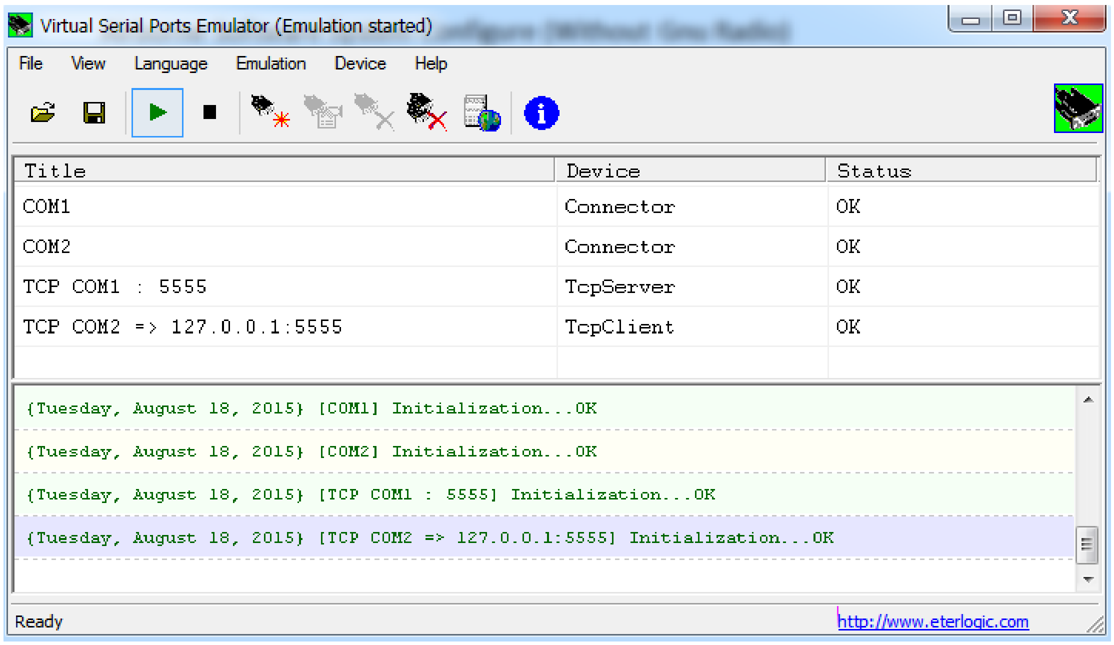
2. Run Simulator on Com1
   NegevSat-Simulator-2015\jar\simulator2 com1 bin.bat
3. Run Airborne Software on Com1
   Make sure that "-timer32 -fast_uart -uart2 //./com1"
   is set in "TSIM2 Configurations-> Extra Options".

2.6.2 Airborne Software System Configure (Without Gnu Radio)

This working mod is used to connect the Airborne Software with simulator and remote computer Ground Station without Gnu Radio

Airborne Software Computer:

1. Configure VSPE in Airborne Software Computer as define below:



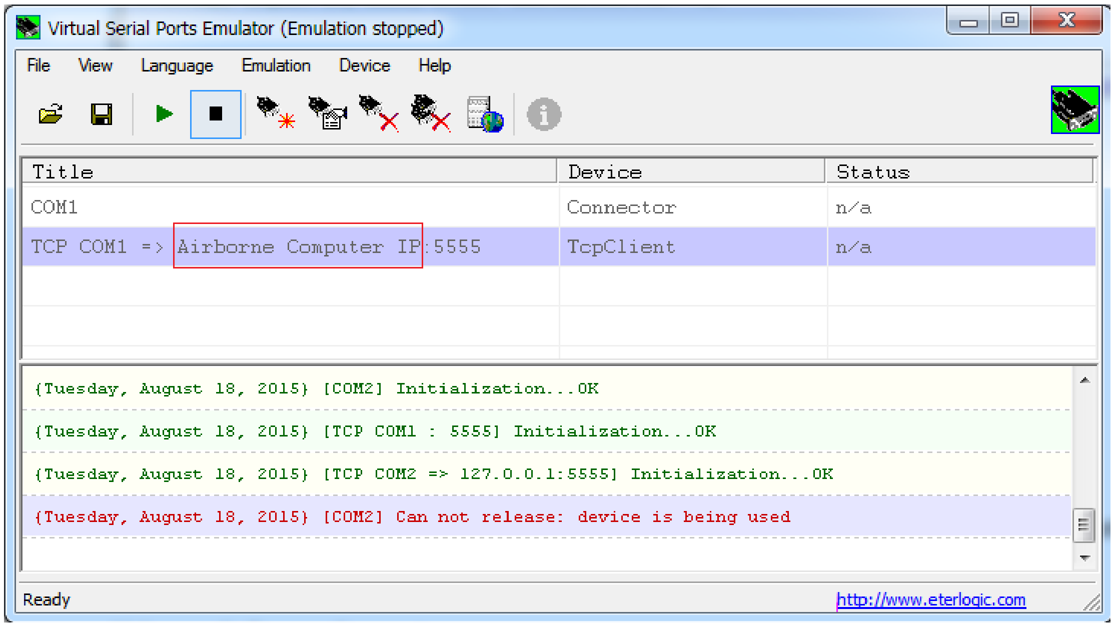
Title	Device	Status
COM1	Connector	OK
COM2	Connector	OK
TCP COM1 : 5555	TcpServer	OK
TCP COM2 => 127.0.0.1:5555	TcpClient	OK

```
{Tuesday, August 18, 2015} [COM1] Initialization...OK
{Tuesday, August 18, 2015} [COM2] Initialization...OK
{Tuesday, August 18, 2015} [TCP COM1 : 5555] Initialization...OK
{Tuesday, August 18, 2015} [TCP COM2 => 127.0.0.1:5555] Initialization...OK
```

2. Run Simulator on Com2
   "NegevSat-Simulator-2015\jar\simulator2 com2 bin.bat".
3. Run Airborne Software on Com1
   Make sure that "-timer32 -fast_uart -uart2 //./com1"
   is set in "TSIM2 Configurations-> Extra Options".

Ground Station Computer:

4. Configure VSPE in Airborne Software Computer as define below:
   Note: set the Airborne Software IP Address



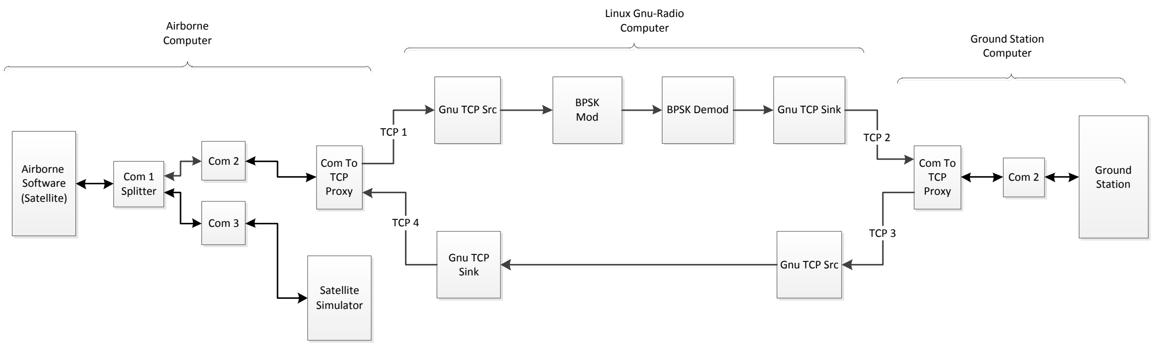
Title	Device	Status
COM1	Connector	n/a
TCP COM1 => Airborne Computer IP:5555	TcpClient	n/a

```
{Tuesday, August 18, 2015} [COM2] Initialization...OK
{Tuesday, August 18, 2015} [TCP COM1 : 5555] Initialization...OK
{Tuesday, August 18, 2015} [TCP COM2 => 127.0.0.1:5555] Initialization...OK
{Tuesday, August 18, 2015} [COM2] Can not release: device is being used
```

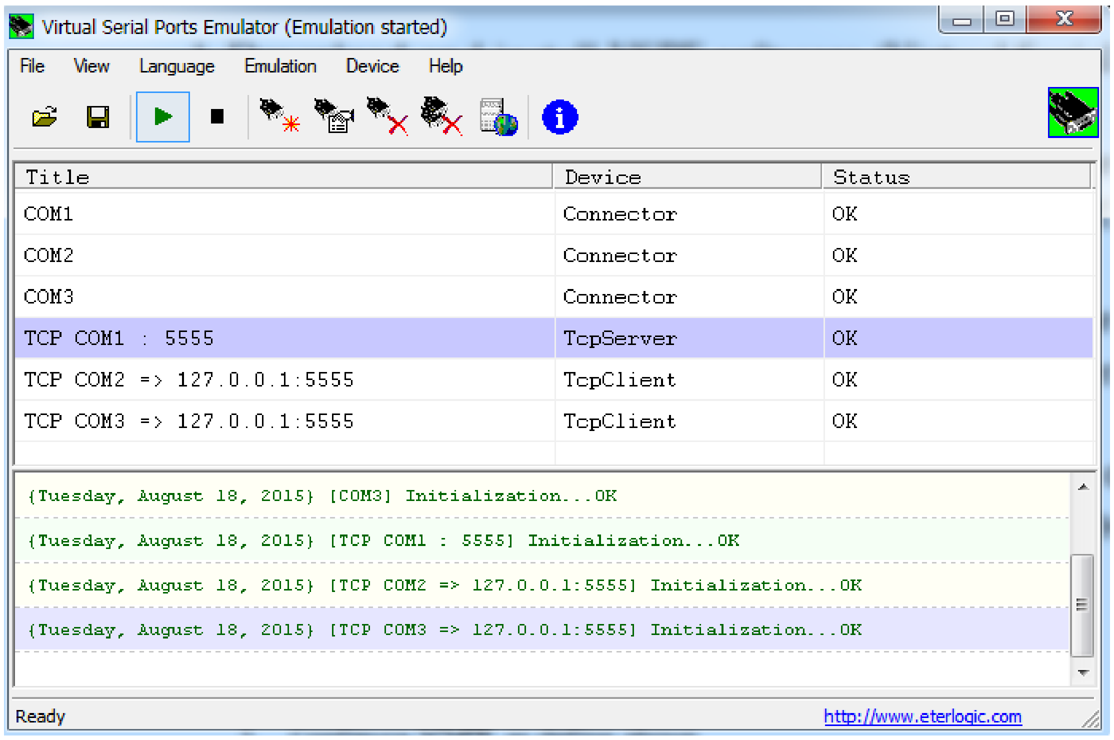
5. Run Ground Station on Com1
   See Ground Station User Guide

2.6.3 Airborne Software Full System Configure

This is the Airborne Software configuration on Full Working mode with Ground Station Gnu Radio and Simulator. For a full video guide and more documents press here.
Full configuration chart:



1. Configure VSPE as define below



Title	Device	Status
COM1	Connector	OK
COM2	Connector	OK
COM3	Connector	OK
TCP COM1 : 5555	TcpServer	OK
TCP COM2 => 127.0.0.1:5555	TcpClient	OK
TCP COM3 => 127.0.0.1:5555	TcpClient	OK

```
(Tuesday, August 18, 2015) [COM3] Initialization...OK
(Tuesday, August 18, 2015) [TCP COM1 : 5555] Initialization...OK
(Tuesday, August 18, 2015) [TCP COM2 => 127.0.0.1:5555] Initialization...OK
(Tuesday, August 18, 2015) [TCP COM3 => 127.0.0.1:5555] Initialization...OK
```

2. Configure Proxy1.bat as following:
   ""%JAVA_HOME%\bin\java.exe" -jar proxy.jar -com COM3 -inip #gnu-ip# -inport 7776 -outip #gnu-ip# -outport 6666"
   Where #gnu-ip# is the ip of the gnu-radio computer.
3. Run Gnu-Radio as described in the Ground Station user guide and in the video.
4. Run Proxy1.bat and proxy2.bat alternately as described in the video.
5. Run Ground Station in Com2 as described in the Ground Station user guide.
6. Run Simulator on Com2
   "NegevSat-Simulator-2015\jar\simulator2 com2 xml.bat".
7. Run Airborne Software on Com1.
   Make sure that "-timer32 -fast_uart -uart2 //./com1"
   is set in "TSIM2 Configurations-> Extra Options".

3. Testing Documentation

3.1 Unit Tests

3.1.1 hardware2test Package:

- **modulesTest package**:
  - AntennaTest tests description:
    - Open antenna and check if the status of the antenna became "open".
    - Close antenna and check if the status of the antenna became "Close".
    - Turn on antenna and check if the antenna became "power on".
    - Turn off antenna and check if the antenna became "power off".

    - Result: All test passed.

  - **AttitudeControlTest tests description:**
    - Create an hardware object, set is attitude to 50 and StartReduceSpinRate() for one work cycle and check if attitude is decrees to 49.
    - Create an hardware object, set is attitude to -10 and StartIncreaseSpinRate() for one work cycle and check if attitude is increase to -9.
    - Result: All test passed.

  - **ThermalControlTest tests description:**
    - Create an hardware object, set is Temperature to 70 and StartReduceSpinRate() for one work cycle and check if attitude is decrees to 69.
    - Create an hardware object, set is Temperature to -20 and StartIncreaseSpinRate() for one work cycle and check if attitude is increase to -19.
    - Result: All test passed.

- **sensorsTest package:**
    - **AttitudeSensorTests tests description:**
        - Set different value in the sensor, and check if the value had been change.
        - Result: All test passed.
    - **CurrentSensorTests tests description:**
        - Set different value in the sensor, and check if the value had been change.
        - Result: All test passed.
    - **DodSensorTests tests description:**
        - Set different value in the sensor, and check if the value had been change.
        - Result: All test passed.
    - **SunSensorTests tests description:**
        - Set different value in the sensor, and check if the value had been change.

      Result: All test passed.
    - **TemperatureSensorTest tests description:**
        - Set different value in the sensor, and check if the value had been change.
          Result: All test passed.

3.1.2 CommunicationTests Package:

We implements boundary values test for this package

- **BinEnergyPacketTests tests description:**
    - Test empty message with no BattaryInfo samples(boundry of num_of_sample = 0)
      Result: the message created successfully and in the right format as defined in the binary protocol with no BattaryInfo samples on it.
    - Test message with one BattaryInfo sample.
      the BattaryInfo sample values are time=0,current=0,volage=0
      Result: the message created successfully and in the right format as defined in the binary protocol with one BattaryInfo samples on it.

- Test message with two BattaryInfo sample.
  the 1st BattaryInfo sample values are time=0,current=0,volage=0
  and the 2nd BattaryInfo sample values are
  time=ulllMax,current=IntMax,volage=IntMax
  Result: the message created successfully and in the right format as defined in the binary protocol with two BattaryInfo samples on it.

- **BinStatusPacketTests tests description:**
  - Test empty message with no ComponentInfo samples(boundry of num_of_sample = 0)
    Result: the message created successfully and in the right format as defined in the binary protocol with no ComponentInfo samples on it.
  - Test message with one ComponentInfo sample.
    the ComponentInfo sample values are componentCode=0,time=0,status=0
    Result: the message created successfully and in the right format as defined in the binary protocol with one ComponentInfo samples on it.
  - Test message with two ComponentInfo sample.
    the 1st ComponentInfo sample values are
    componentCode=0,time=0,status=0
    and the 2nd ComponentInfo sample values are
    componentCode=charMax,time=ullMax,status=byteMax
    Result: the message created successfully and in the right format as defined in the binary protocol with two ComponentInfo samples on it.
- **BinTempsPacketTests tests description:**
  - Test empty message with no TempSample samples(boundry of num_of_sample = 0)
    Result: the message created successfully and in the right format as defined in the binary protocol with no TempSample samples on it.
  - Test message with one TempSample sample.
    the TempSample sample values are time=0,tempature=0
    Result: the message created successfully and in the right format as defined in the binary protocol with one TempSample samples on it.
  - Test message with two TempSample sample.
    the 1st TempSample sample values are time=0,tempature=0
    and the 2nd TempSample sample values are time=ullMax,tempature=intMax
    Result: the message created successfully and in the right format as defined in the binary protocol with two TempSample samples on it.

- **CMDParser (command parser) tests description:**
  - Create a valid string which contains xml with missions to perform, construct XML which represents the string and try to parse it into readable work objects, the test asserts that the works which were parsed are consistent to the pre-defined missions.

  Result: All works were parsed correctly.
- **Validator tests description:**
  - Valid XML test – create a valid missions string, construct XML using the Validator and assert that validation returns “true”.

  Result: validation returned “true” – test pass.
  - Bad XML syntax test - create a non valid missions string – incorrect xml syntax, construct XML using the Validator and assert that validation returns “false”.

  Result: validation returned “false” and printed the cause of the failure to the screen.
  - XML doesn’t fit the protocol test – create XML which doesn’t fit the data protocol (valid xml syntax) and assert validation returns “false”.

  Result: validation returned “false” and printed the cause of the failure to the screen.
- **Send and receive tests description:**
  - Send test – create a valid downstream packet (with telemetry), tries to send the data via the communication channel, asserts that the data was sent.

  This test requires the user to connect serial cable (or virtual port) and to configure the TSIM2 emulator.

  Result: The data was sent successfully and send method returned true – this can also be done using “TeraTerm” terminal by seeing the data arrives to the terminal.
  - Receive test – This test requires the user to connect serial cable (or virtual port) and to configure the TSIM2 emulator. Also the user should write data to the serial port, the test receives the data using the communication handler and asserts that the data arrives.

  Result: the data arrived and was printed to the screen.
- **TLMParser (Telemetry Parser) tests description:**
  - Creates static, energy and temperature packets, using the TLMParser (in the same way as in the LifeCycle task), then converts the packets to strings and asserts that the packets were created as expected.
- **WorkQueue tests description:**
  - Basic test – creates 4 different kinds of work with different time of execution. Enqueues the works into the WorkQueue. Sorts the queue, dequeues all works and asserts that the works were sorted as expected.
  - Overflow test – enqueues works until the queue is overflowed, then enqueues one more work and asserts that the oldest work was overwritten by the last work.
- **SendReceiveQueue tests description:**

- Basic test – enqueues 5 messages (data), then dequeues all messages and asserts that messages were stored in FIFO order.
- Overflow test – enqueues messages until the queue is overflowed, then enqueues one more messages and asserts that the oldest message was overwritten by the last message.

3.2 Integration Tests:

- System tests are performed using the simulator – a Java application which connects to the system using serial port communication and interacts with the system (instructions for running the simulator can be found in section 2.4).

- The simulator simulates the main functionality of the system by executing end to end use cases (several units combined).
  When running the simulator a detailed menu with all kinds of operations appears, in order to test a single use case simply type the required key (as described in the menu) and observe the behavior of the system (see logs and state of the satellite).

- The simulator makes a role of a "driver" and replaces the Ground Station.

3.3 System Tests:

- In the system tests we are connecting a full system configuration including the Ground Station, the Communication Device (Gnu Radio), the airborne software and the simulator.
- In this kind of test the simulator takes a role of fault and event maker
  Its simulate an events like temperature drop down or spin rate became high and likewise.
- In the System Tests we test all the requirements as described in the previous documentations.
- These tests are run manual because of the complication of making a automatic tests over 3 different computers with lots of infrastructure software (like VSPA TSIM and Ext…)