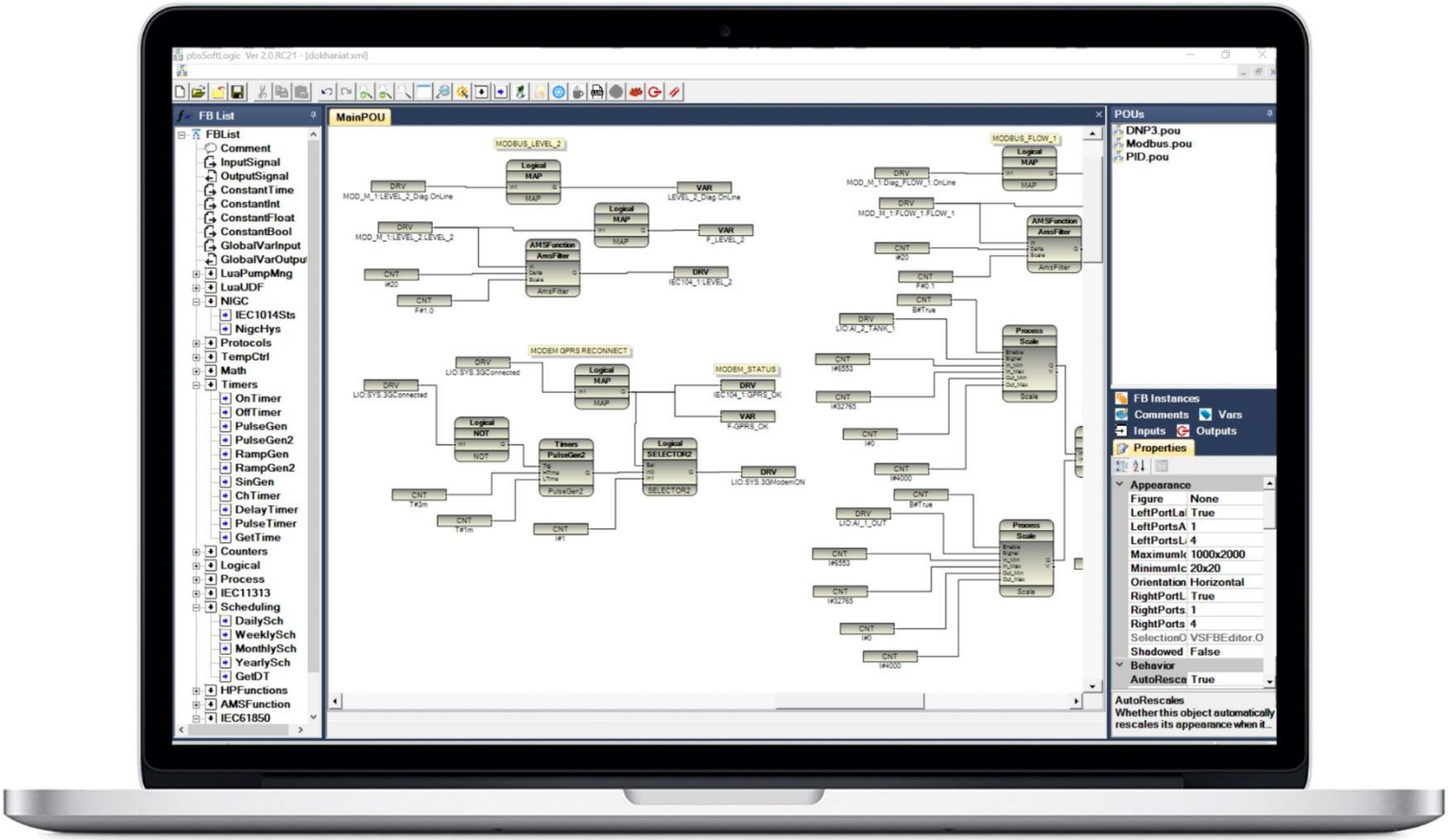
pbsSoftLogic

Concepts



1 – Introduction

pbsSoftLogic is an open RTU/PLC programming environment from pbsControl LTD. pbsSoftlogic is developed by Dot Net Technology. The development version of pbsSoftLogic runs on the Windows operating system, and its runtime runs on any Linux-based controller.

pbsSoftLogic has the following specifications:

- Standard function block programming environment

- Lua programming and C language are used to develop user defined function blocks

- Offline simulation of the program developed in Windows

- More than 400 ready-made and tested function blocks for easy programming.

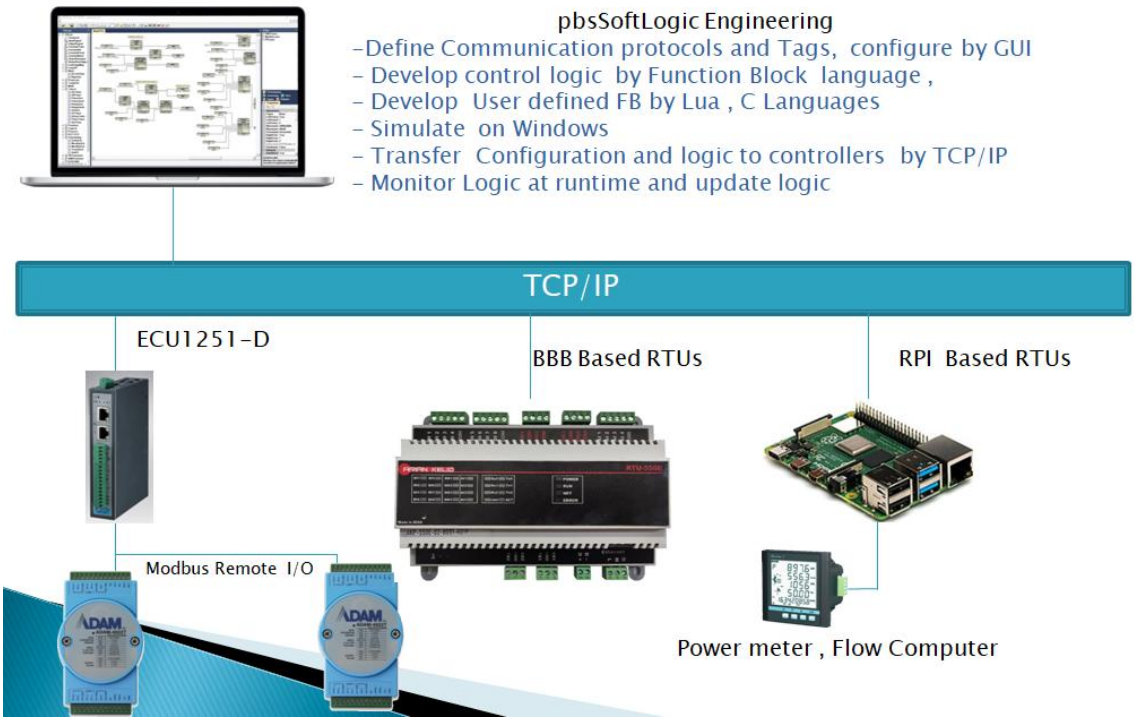
- Support Modbus, DNP3, IEC101/104, OPC UA, Redis Client, MQTT, SQL Client, GSP, IEC62056-21 drivers

- IEC62351 is supported for DNP3 and IEC104 protocols.

- Supports program monitoring and forcing variables during execution.

- RTU status monitoring tool

- The runtime kernel runs on any Linux-based controller. The Windows CE, win32, and QNX runtime kernels have been removed in version 5.0.


pbsSoftLogic Engineering
-Define Communication protocols and Tags, configure by GUI
- Develop control logic by Function Block language ,
- Develop User defined FB by Lua , C Languages
- Simulate on Windows
- Transfer Configuration and logic to controllers by TCP/IP
- Monitor Logic at runtime and update logic
TCP/IP
ECU1251-D
BBB Based RTUs
RPI Based RTUs
Modbus Remote I/O
Power meter , Flow Computer

2 – pbsSoftLogic installation

pbsSoftLogic IDE runs on Windows operating system which supports Dot Net Frame work 4.7.2.

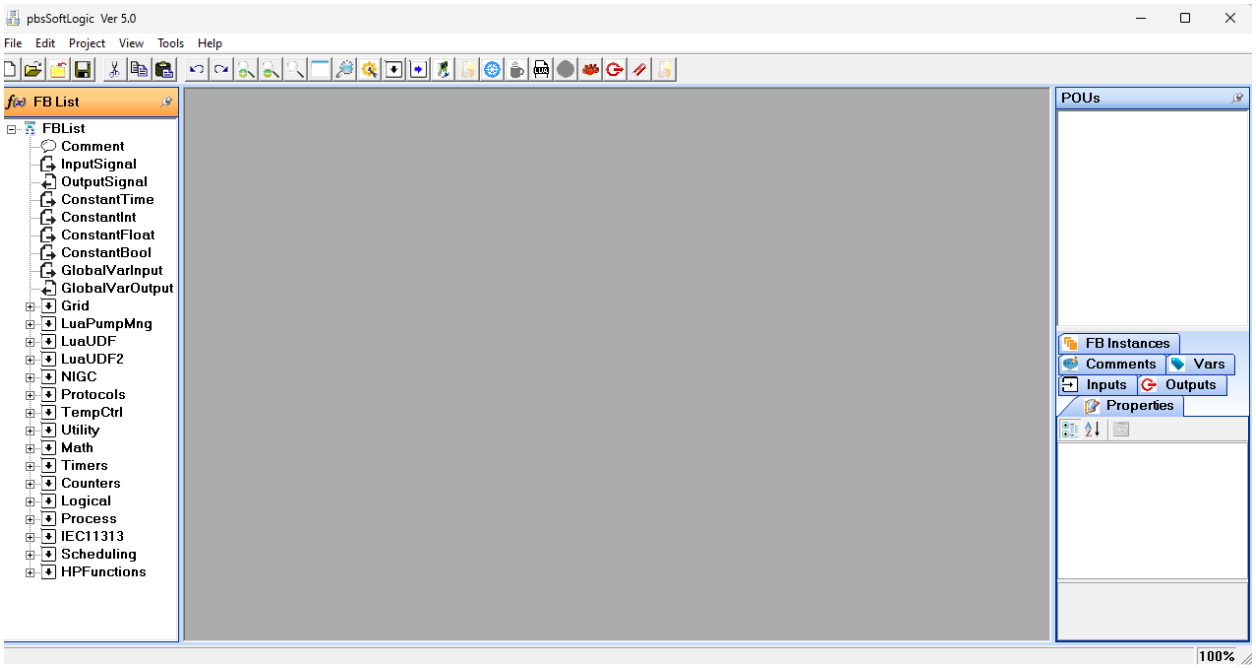
You can download the latest pbsSoftLogic IDE from http://www.pbscontrol.com

Simply extract the file and run pbsFBEditor.exe. No need for any installation process

Do not copy the pbsSoftLogic folder to the path with special characters such as space, /, (,).

In this document, we assume that you have copied pbsSoftLogic to the c:/pbsSoftLogic folder.

When you run pbsFBEditor.exe you will see the following screen:



3 – Basic concepts

Writing programs for SCADA systems is a critical task. Using low-level languages such as C/C++ and C# for such projects is not recommended for the following reasons:

1. Non-reusable
2. Difficulty in transferring the project to others and training other engineers to continue the project
3. High risk during execution for stability and error-free
4. Taking a long time to develop the project

Function Block language is a language for control engineers. They can focus on process logic without worrying about the software side. FB is a complete graphical language with many tested and ready-made functions inside.

Using the function block language has the following advantages:

1 –100% reusable .There are many tested and ready-made functions that can be used in different projects with complete documentation.

2 – It is very easy to train control and process engineers for programming in projects.

3 – pbsSoftLogic has been used in many projects and sites for the last few years, so there is no error in the runtime and development environment.

4- You can use pbsSoftLogic and Function block language as a framework for your entire automation projects.

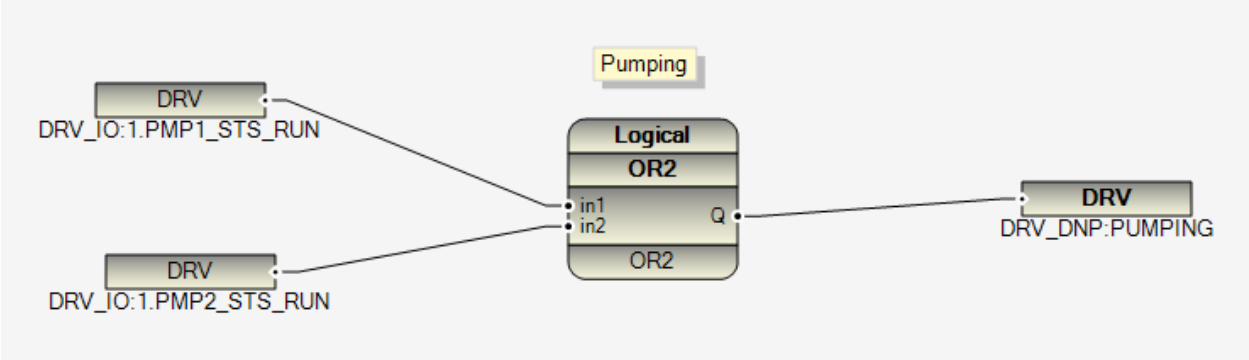
pbssoftLogic is an IDE for developing Function Blocks, simulating and downloading programs on Linux based controllers. You can use Lua Scripting or C language to develop new FB and integrate with IDE.

All pbsSoftLogic FB source code is open source and located in \FBDSourceCode.

4 – Function Block Programming Language

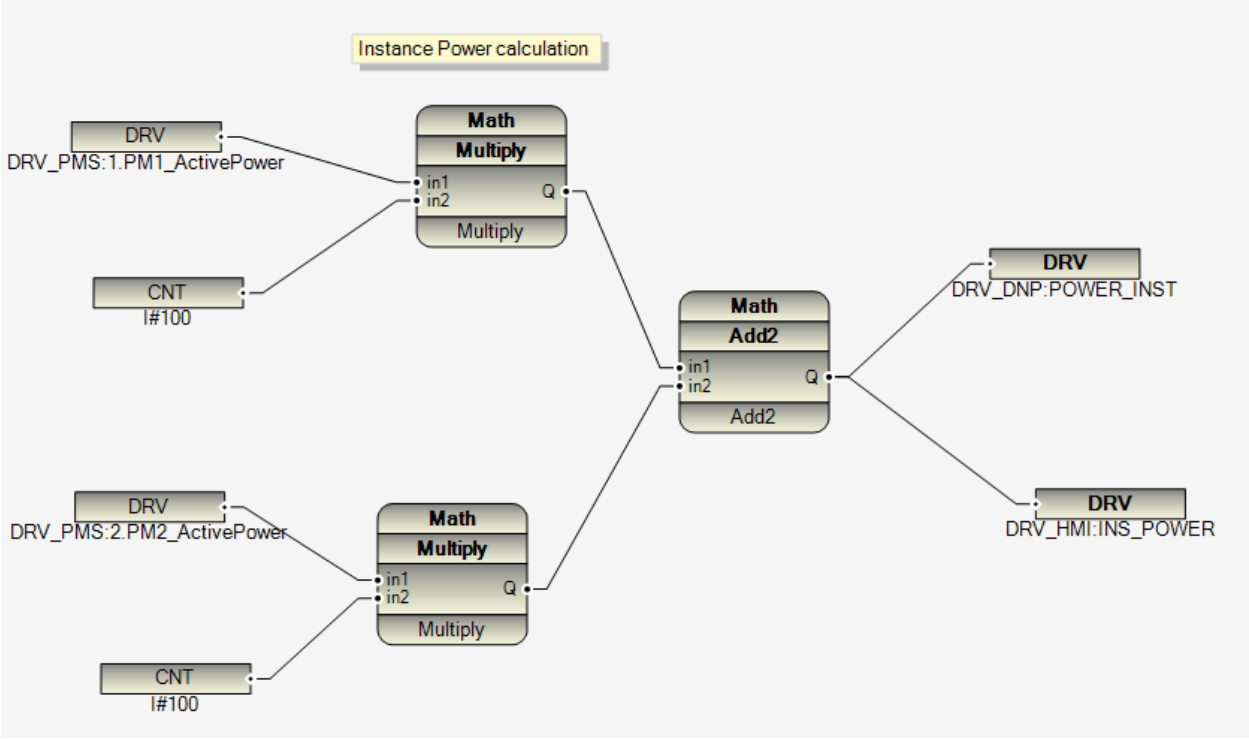
The main element of a program is the FB (Function Block). Below you can see some simple examples.

Example1:



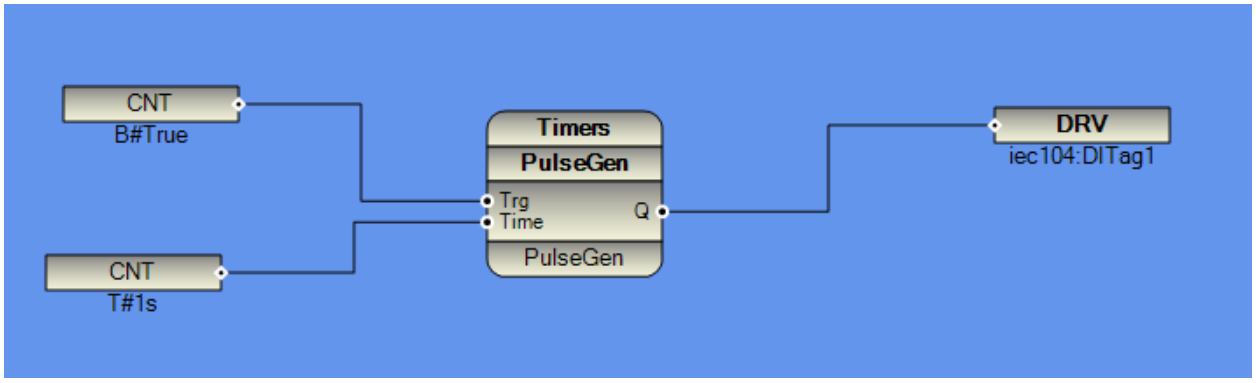
In this logic, two signals PMP1_STS_RUN and PMP2_STS_RUN are input to the OR FB and the output is written to the PUMPING signal.

Example 2:



In Example 2, PM1_ActivePower is multiplied by 100, PMP2_Activepower is multiplied by 100, and both results are added together and written to the Power_Instance signals. (write on two different drivers).

Example 3:



In example 3, a pulse with a period of one second is generated and written to the DTTag1 signal in the iec104 driver.

The main elements of a functional block program are:

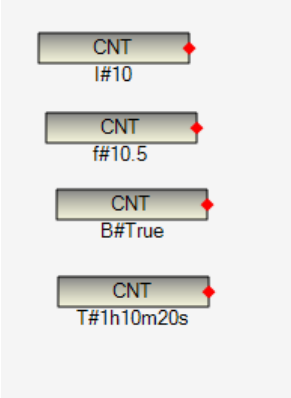
1 – Input/output signals: connected to communication drivers and local I/O.

2 – FB: Function Blocks.

3 – Connection between I/O signals, FBs and between FBs.

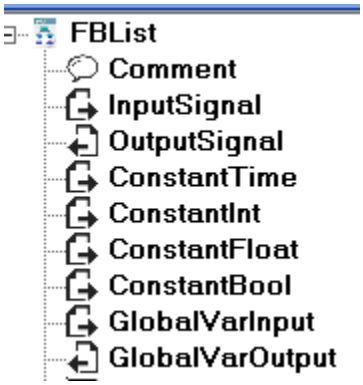
4 – Constant signals: different type of Constant Signals: Integer (I), Float (F), Boolean (B), Time (T)

Constant Signal Format: Type # Value.

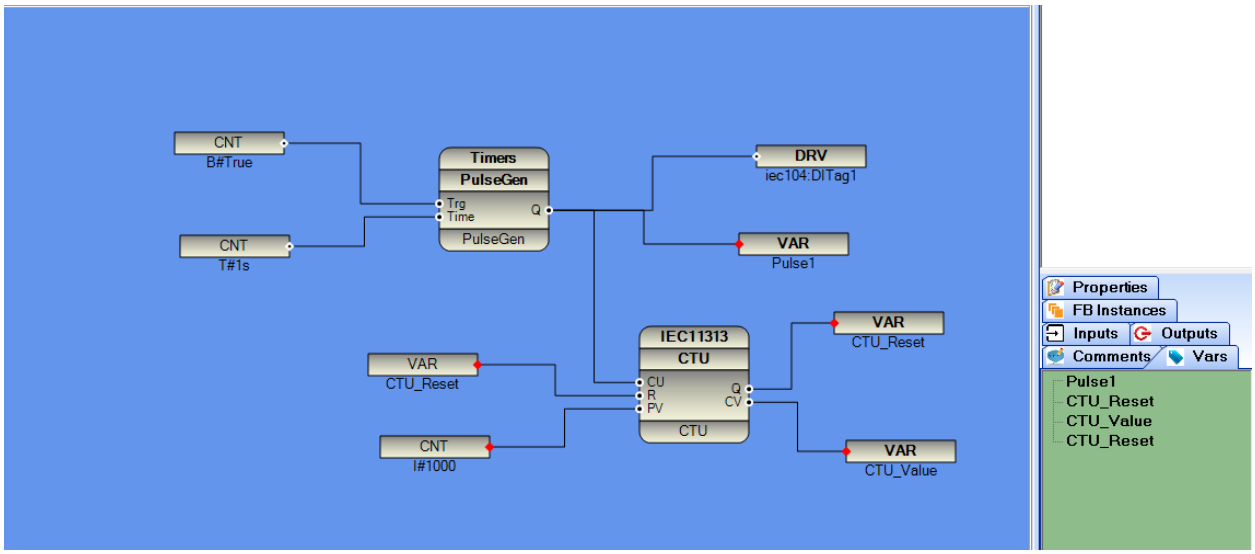


5- Internal variables: unlimited internal variable is possible in logic, but each instance must have a different name. Variables with the same name in the program have the same value.

From FBList you can select Constant Tags, Global Variables or Driver tags.



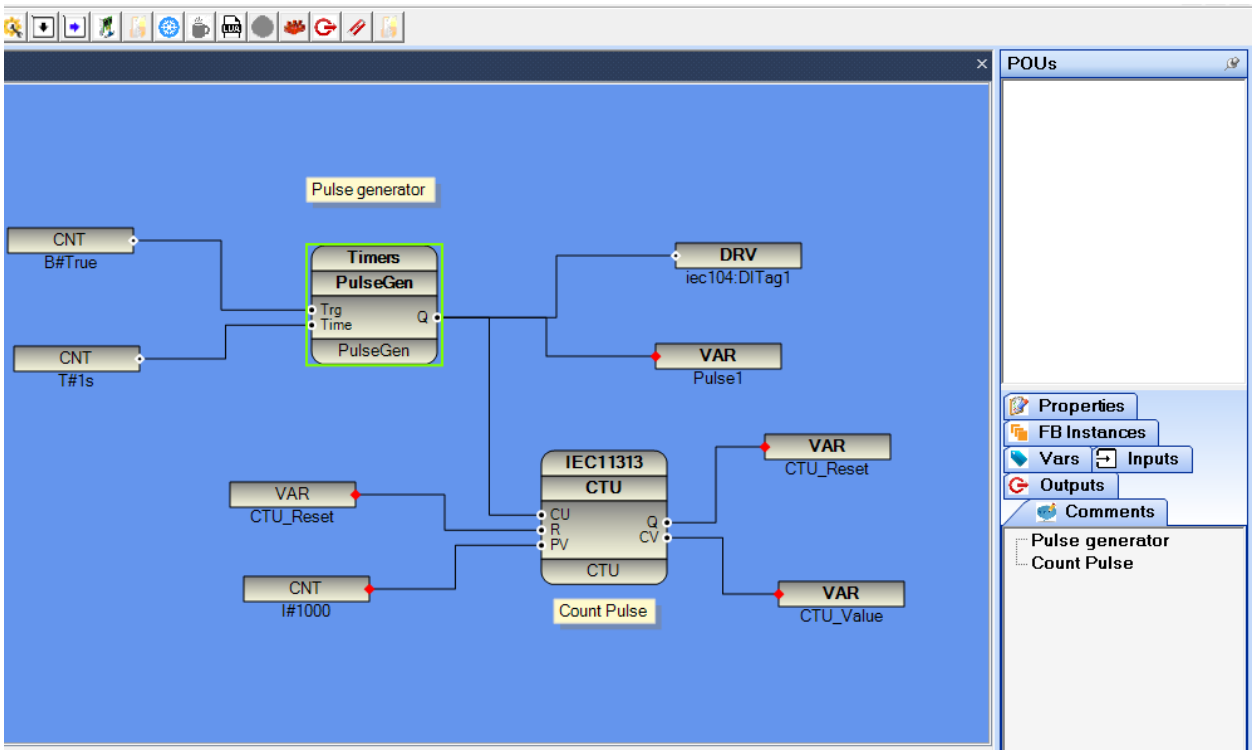
In the following example you can see constant and global variables usage.



In the Vars tool, you can see a list of all global variables in the open program.

By double-clicking any variable item in the Vars tool, Logic will focus on the variable's location. So you can easily browse and check all the variables.

6 - Comments: You can put a comment anywhere in the logic. Drag a comment element from the FBList and drop it into the logic. Then click on the comment and change its content. A comment is like a dynamically sized yellow text box.



You can see the list of all comments in the open application from the Comments tool. You can view the location of the comment by double-clicking on the comment list.

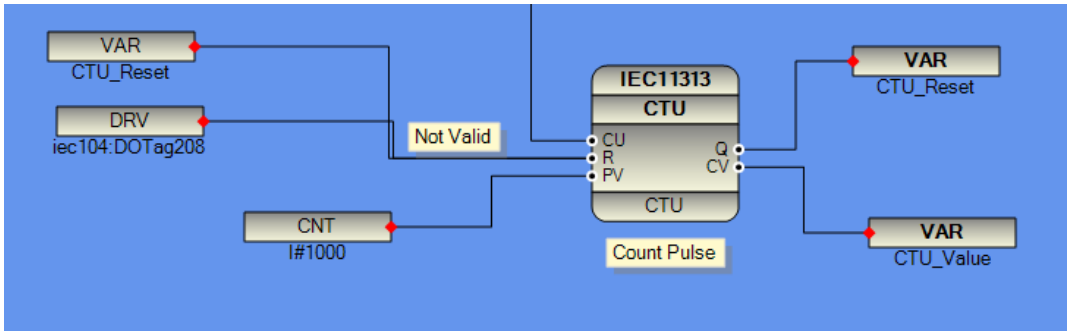
Function block programming rules:

1 - FB inputs (left) are always connected to a source. You can connect a source (I/OSignal, Internal Variable and Constant) to different FB inputs. But multiple source to one FB entry is not valid.

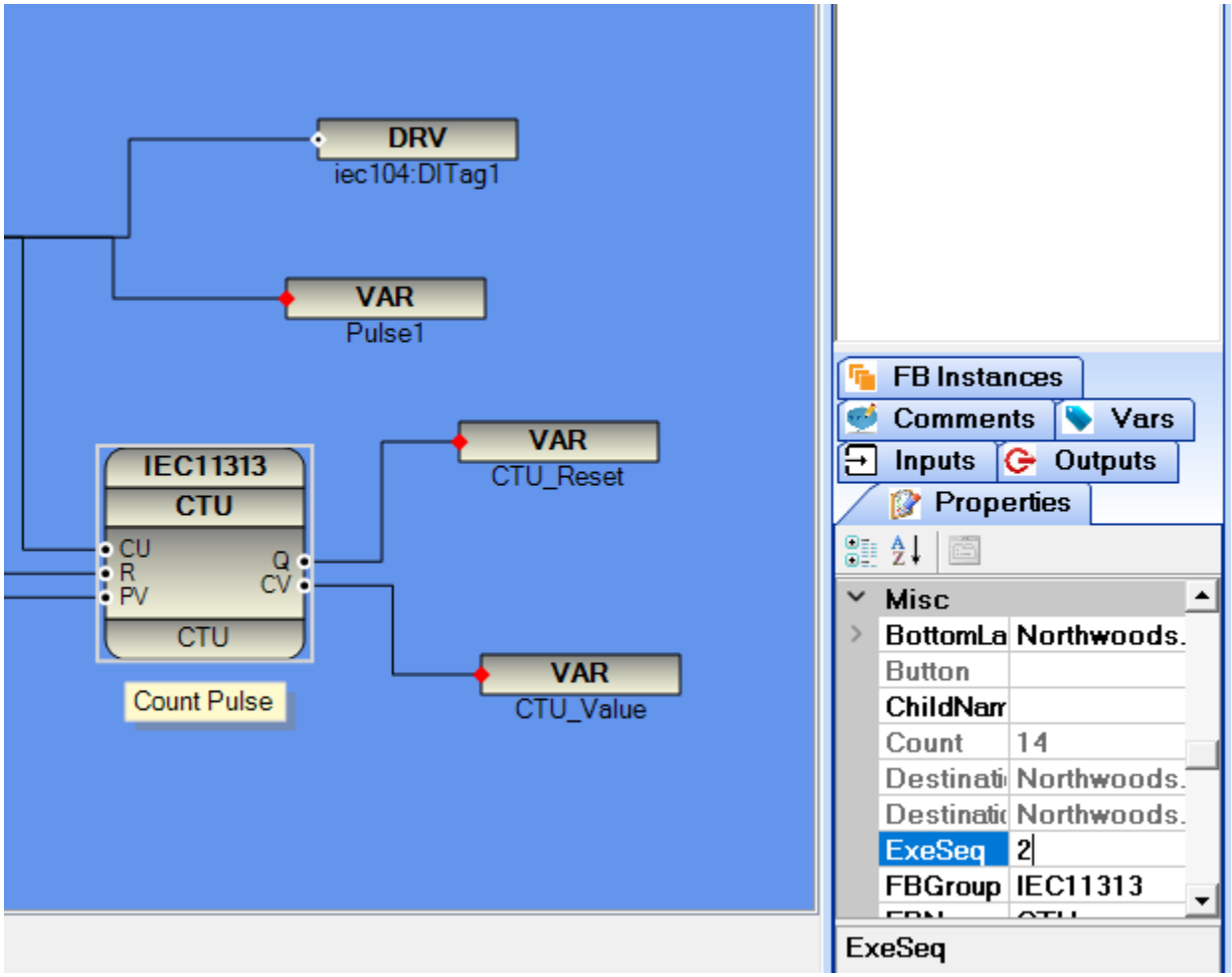
2 - FB outputs (right side) can be connected to different signals. (not fixed signals)

3 - There is no limit on the number of levels of FB connections.

In the image below you can see an invalid connection for the CTU Function Block, Reset input. You cannot bind multiple variables to one FB input.



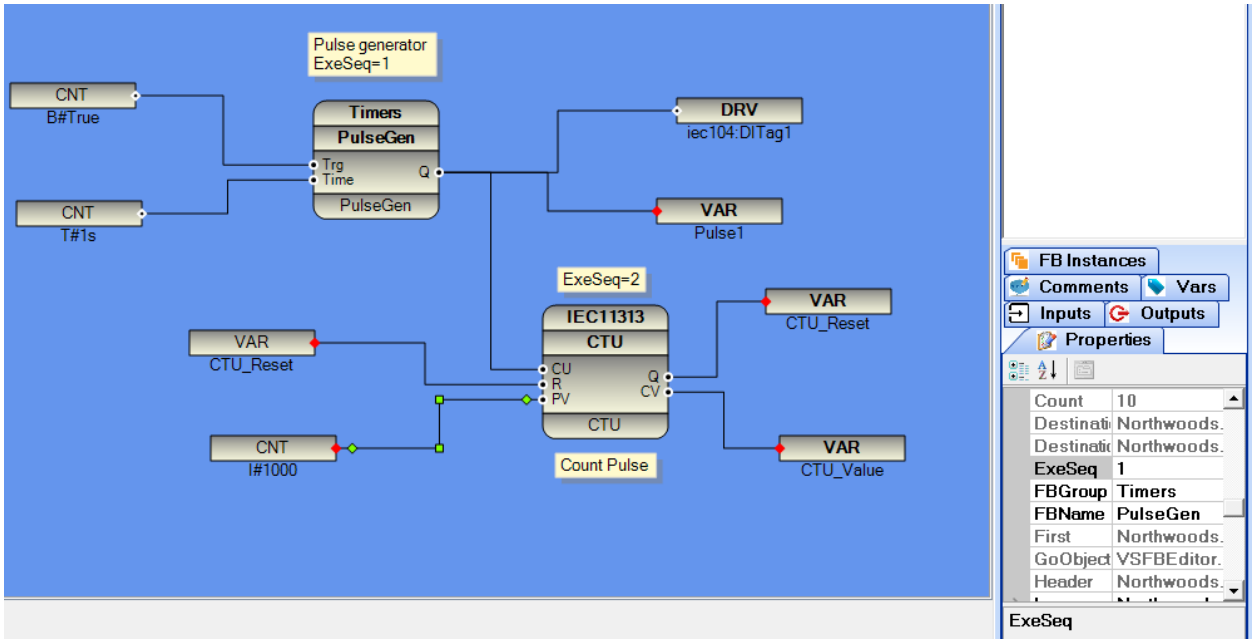
4 - Logical execution: each FB has an execution number. Click FB and press F4, you can see FB properties window. Scroll through the properties to find ExeSeq.



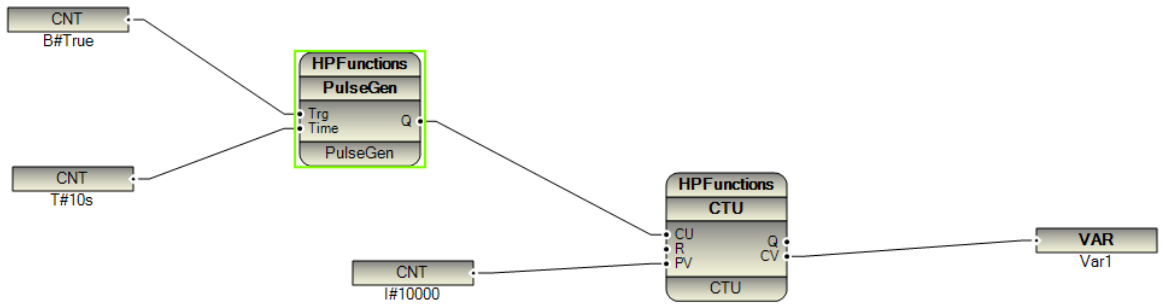
When you start developing logic, FBEditor automatically increments the ExeSeq value for each FB you use. But you can change its value and in this way you can control how the logic is executed. We recommend that you set all ExeSq numbers manually, because when you copy-paste part of the logic, FBEditor will put the same values for the pasted elements. FBEditor sorts all Fbs based on ExeSeq number and compiles and generates output file based on ExeSeq command.

In the figure below, PulseGen has ExeSeq number equal to 1 and CTU has ExeSeq number equal to 2.

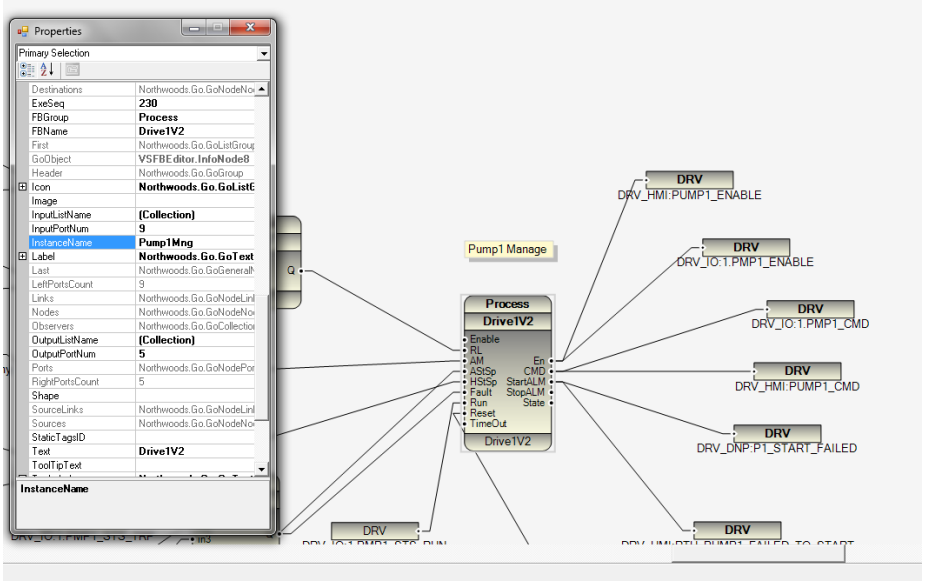
Therefore, at runtime, PulseGen is executed first and then CTU.



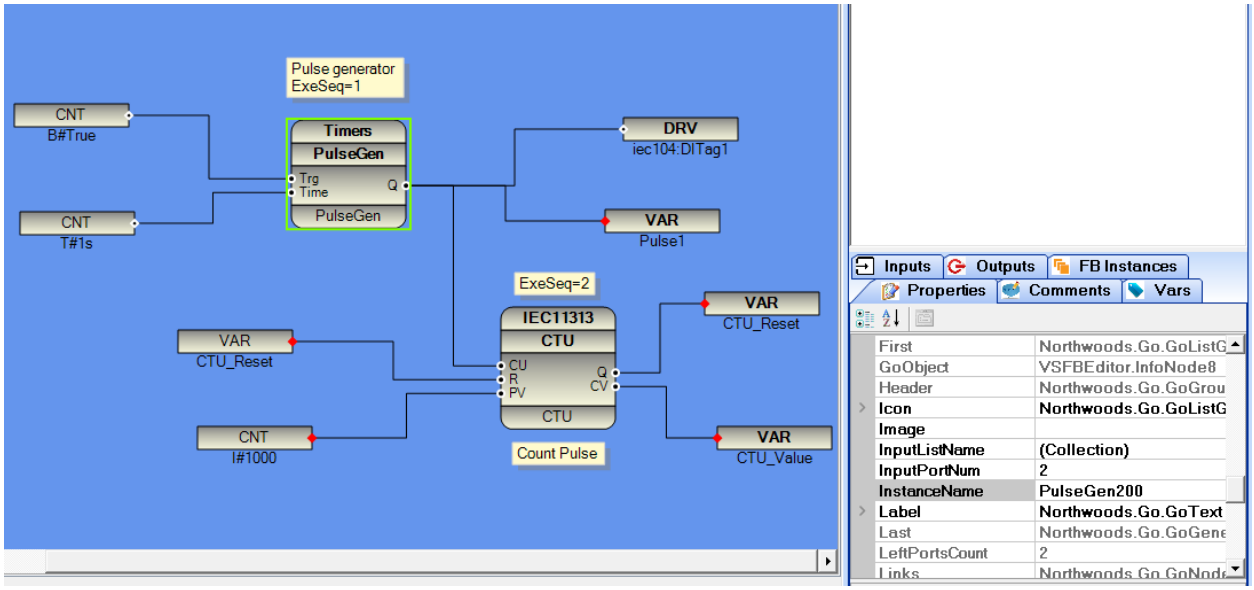
In the following example, the ExeSeq value for PulseGen is greater than CTU, but the logic resolves without problems and there is no output for CTU FB in only one RTU cycle.

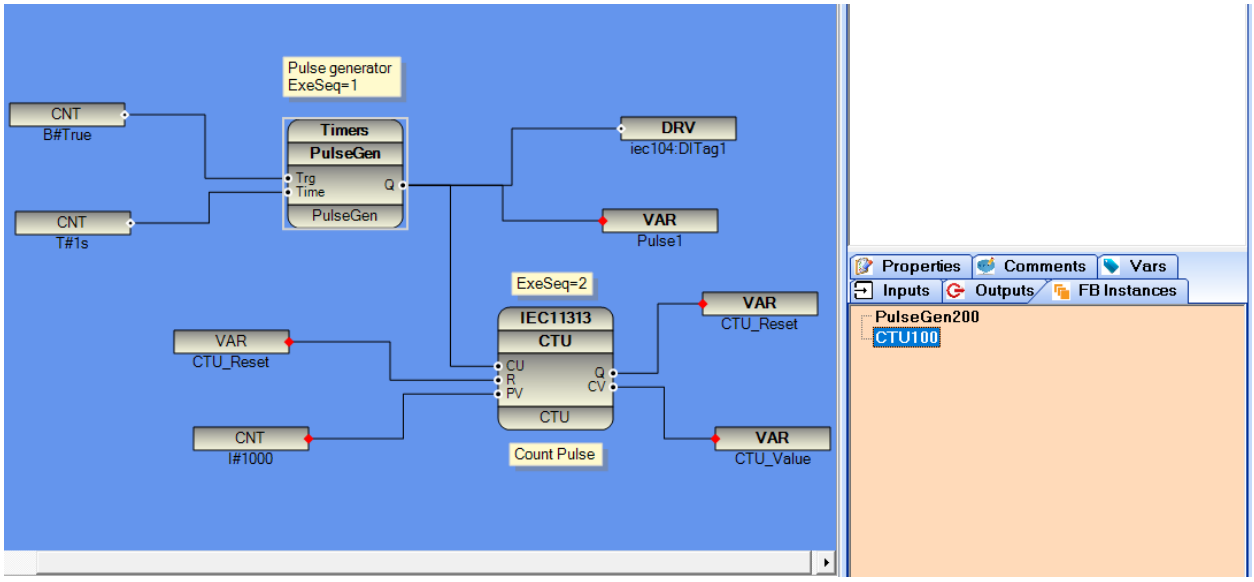


5 - Logic FB Instance name: Each FB has FBName and instance name. These two properties are equal by default. But you can change the instance name to any unique name in your logic. Suppose you are controlling a pump by Drive1V2 FB. By renaming the FB instance to "Pump1Mng", the compiler will use Pump1Mng as the compile-time identification of the FB. By default it uses the PartID attribute, which is always unique in logic.



In the figure below, we change the name of the PulseGen instance to PulseGen200 and the CTU to CTU100, then you can move the logic through the FB Instance window by double-clicking on the instance names.





In the runtime kernel, the PartID is used as the unique identifier for each FB to manage static data for each FB.

When you make warm update for the logic, because the PartID is assigned by the compiler at compile time, so a different PartID can be assigned to the same FB. But if you use the instance name, the runtime kernel will always use the instance name to handle static FB data, so ensures proper warm update at runtime.

Note: If you want 100% warm logic update, you must use instance name for critical function blocks.

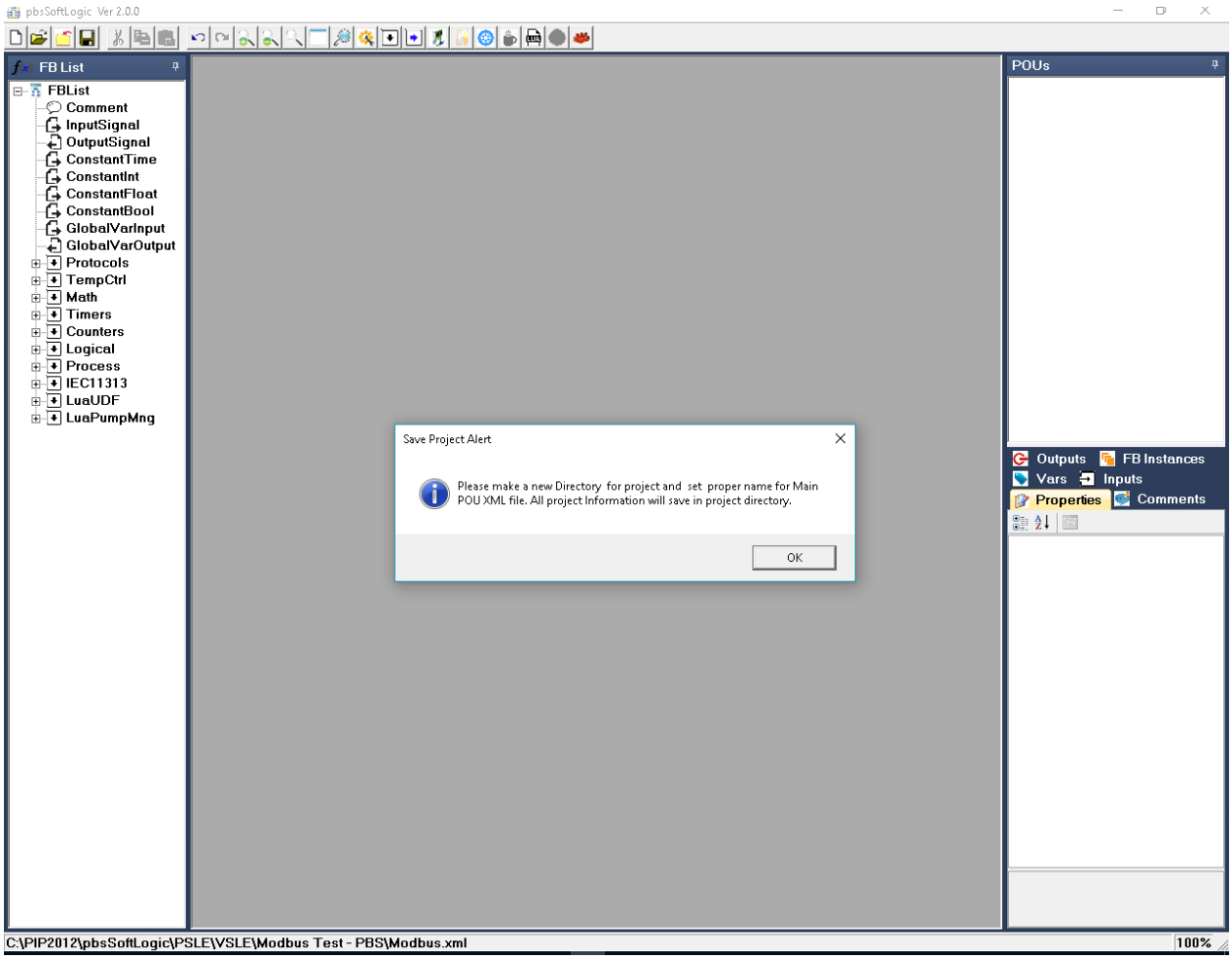
6 - You can write your logic in several POUs (Program Organization Unit). All POUs are merged into one file based on ExeSeq number and passed to the runtime kernel. Therefore, multiple POUs are more suitable for structured programming, and there is no difference in runtime if you write all programs in the main POU or split it into multiple POUs.

7- Logic execution always starts from FB with the smallest ExeSeq number. First, all input signals are read and transferred to the FB, the logic of the FB is solved, and then all outputs are written to the output signals. Then the next FB is executed. Note that the FBs for ExeSeq are always sorted in ascending order in the final program file transferred to the execution core.

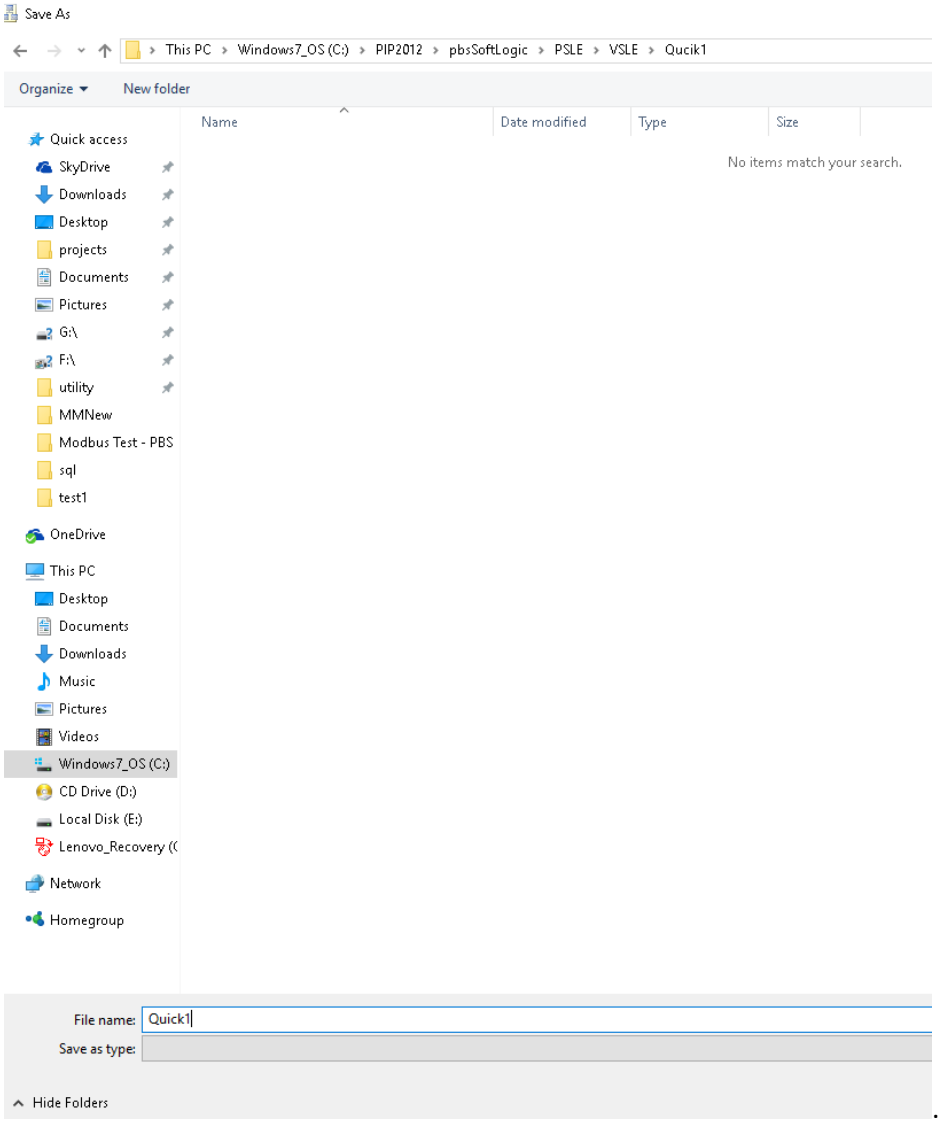
5 – Quick Startup

In this part, we will write a simple logic with pbsSoftLogic and run on Linux controller.

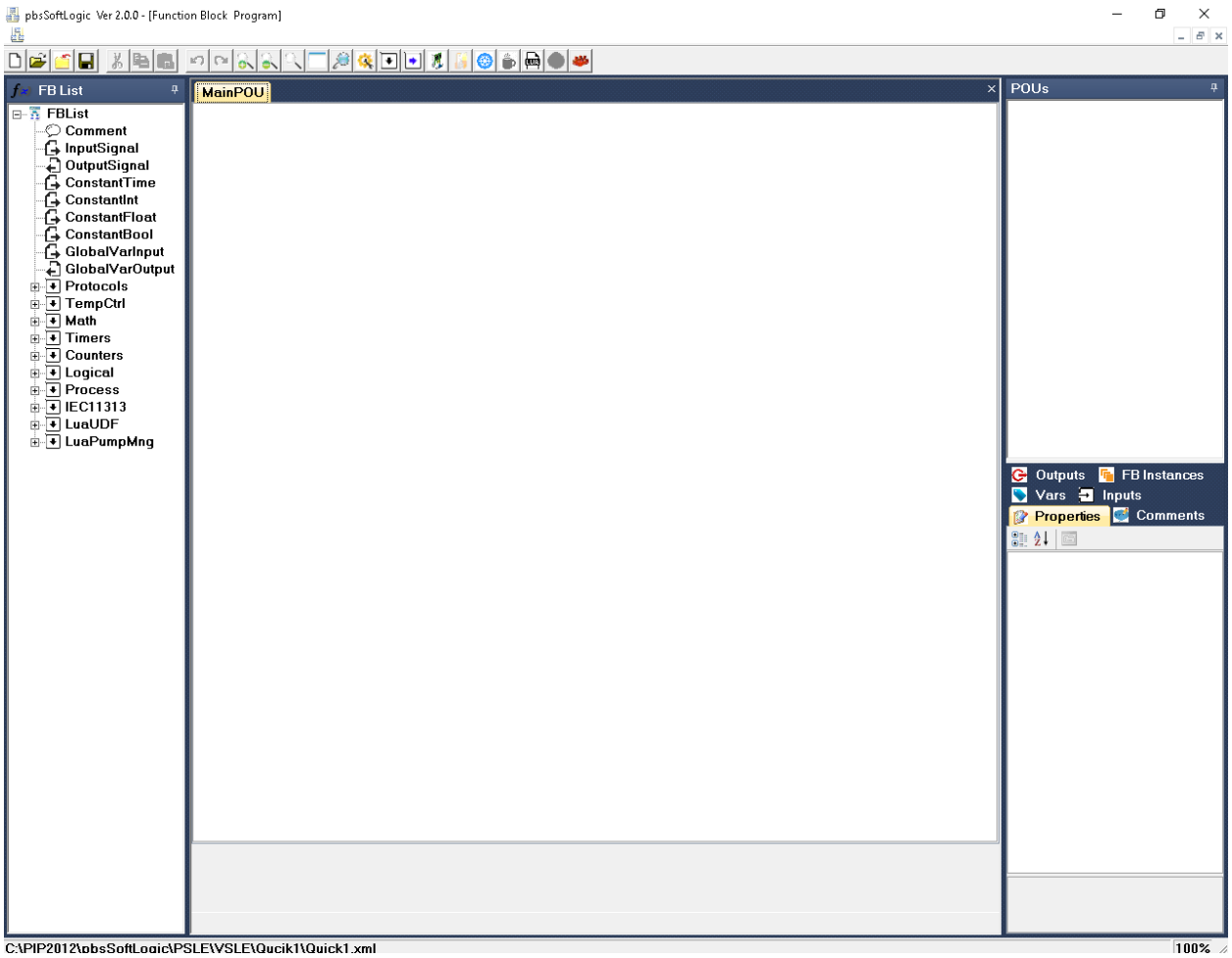
Step1: Make a new Application with pbsSoftLogic . Run pbsFBEditor.exe . In File Menu, Select New.



At first step you should make a directory for your project. You can make anywhere in your system with proper name related to your project. Suppose we will make Quick1 Directory in pbsSoftLogic VSLE Path.



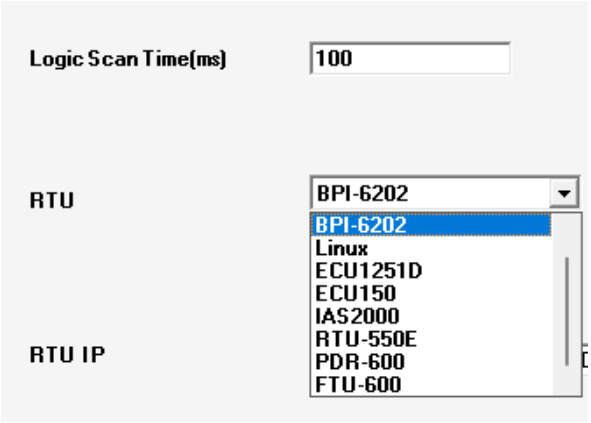
Click the Save button, the save form will automatically close and you are ready to configure and program. You could not close the MainPOU and it will remain open as long as the project is open.


pbsSoftLogic Ver 2.0.0 - [Function Block Program]
FB List
FBList
Comment
InputSignal
OutputSignal
ConstantTime
ConstantInt
ConstantFloat
ConstantBool
GlobalVarInput
GlobalVarOutput
Protocols
TempCtrl
Math
Timers
Counters
Logical
Process
IEC11313
LuaUDF
LuaPumpMng
MainPOU
POUs
Outputs
FB Instances
Vars
Inputs
Properties
Comments
C:\PIP2012\pbsSoftLogic\PSLE\VSLE\Qucik1\Quick1.xml
100%

Step2 : Select your RTU Type . Click on project setting button 

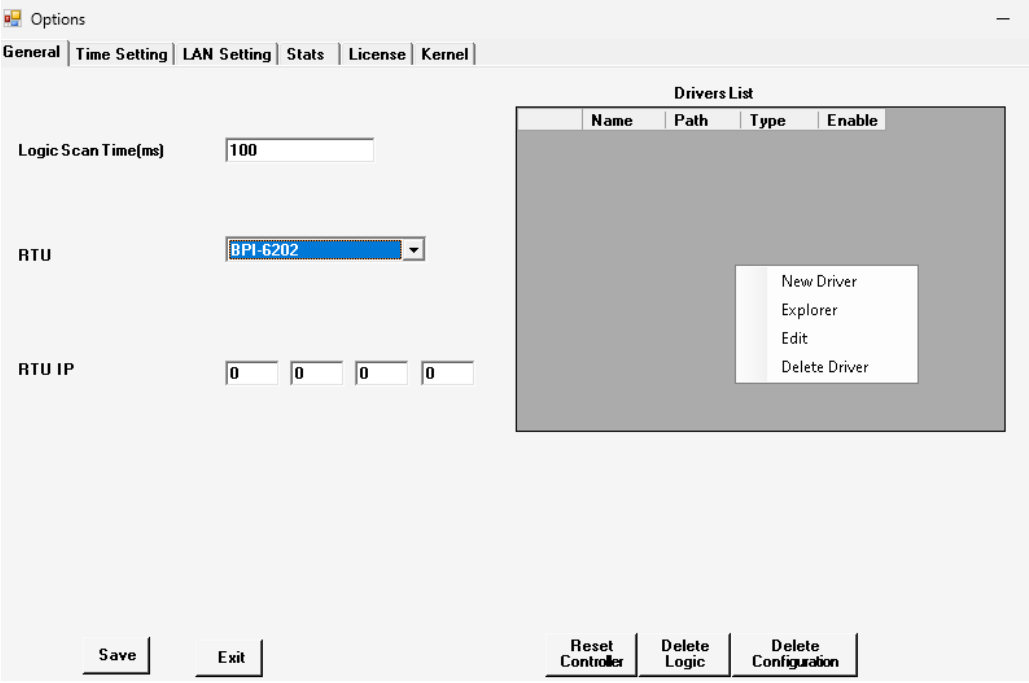
When the project settings page opens, "GeneralBeagleBone" is displayed as the controller by default.

Open the "RTU" combo box, you can see the different RTUs supported by pbsSoftLogic. For now, use the same "BIP-6202". Enter the IP of the RTU, for example 192.168.1.164. Keep the logic scan time at 100ms. This is the loop time to read all the driver inputs, solve the logic and write the driver outputs. Every 100ms the above sequence will repeat, but the whole sequence may only take 2ms to complete and the processor will be in sleep mode for 98ms. Click the Save button to save the settings.



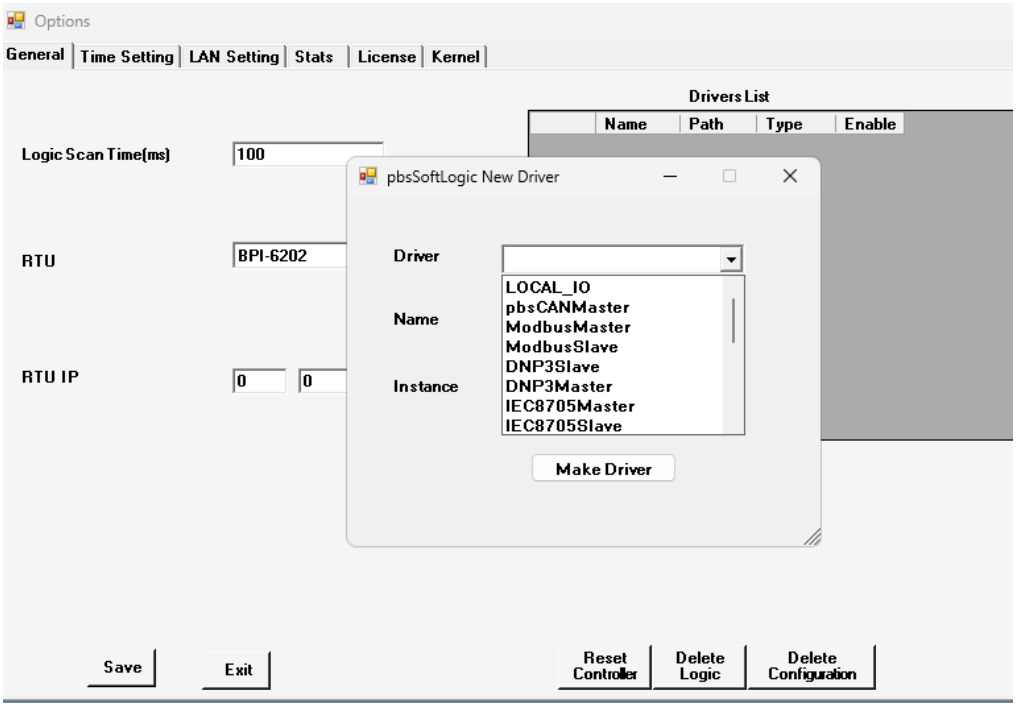
Step3: Define Local IO for RTU

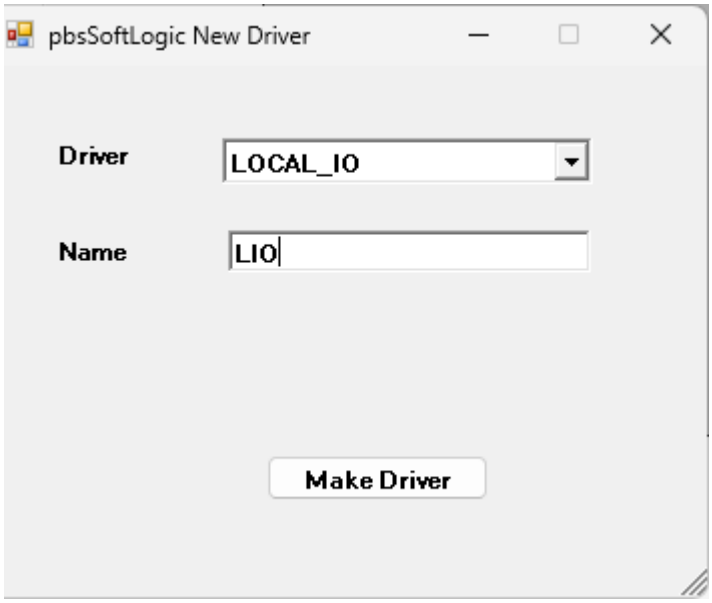
Right-click in the list of drivers and select the New Driver command:



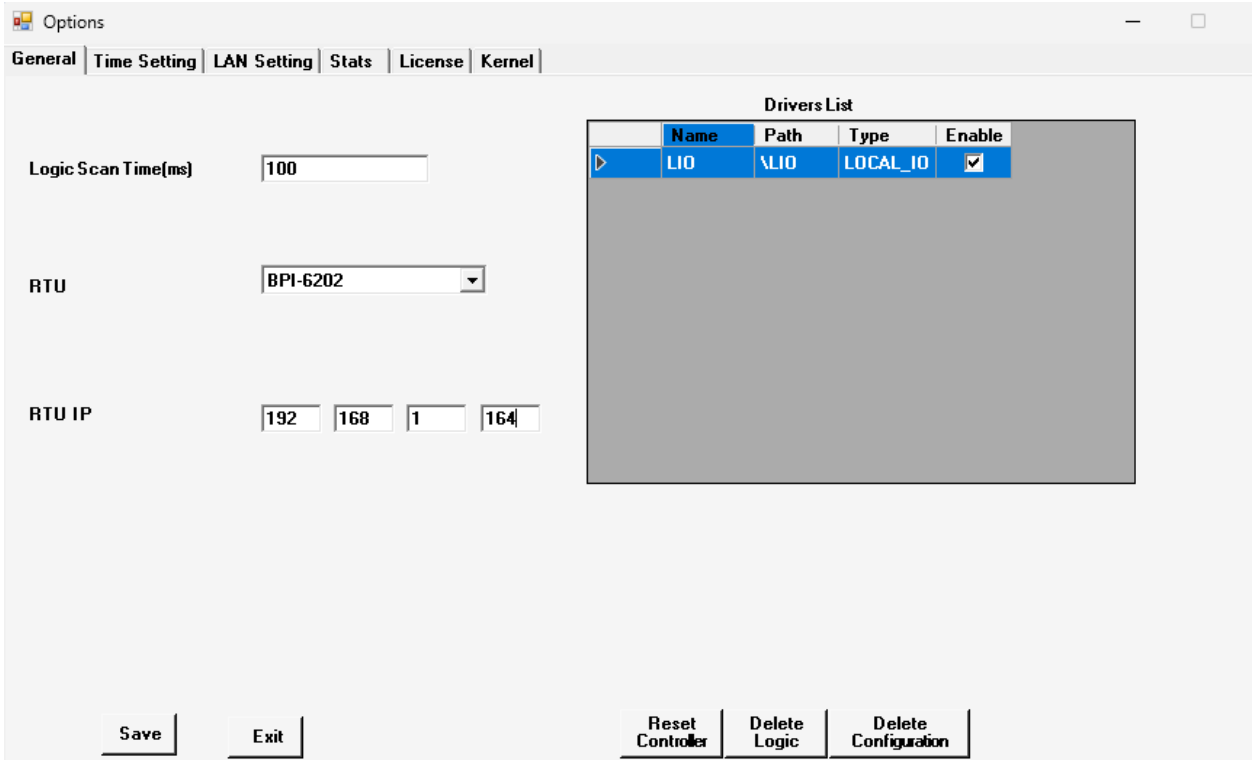
It will show list of Supported Drivers of pbsSoftLogic :

Select Local _IO and choose a unique name for the driver. (For example, LIO)



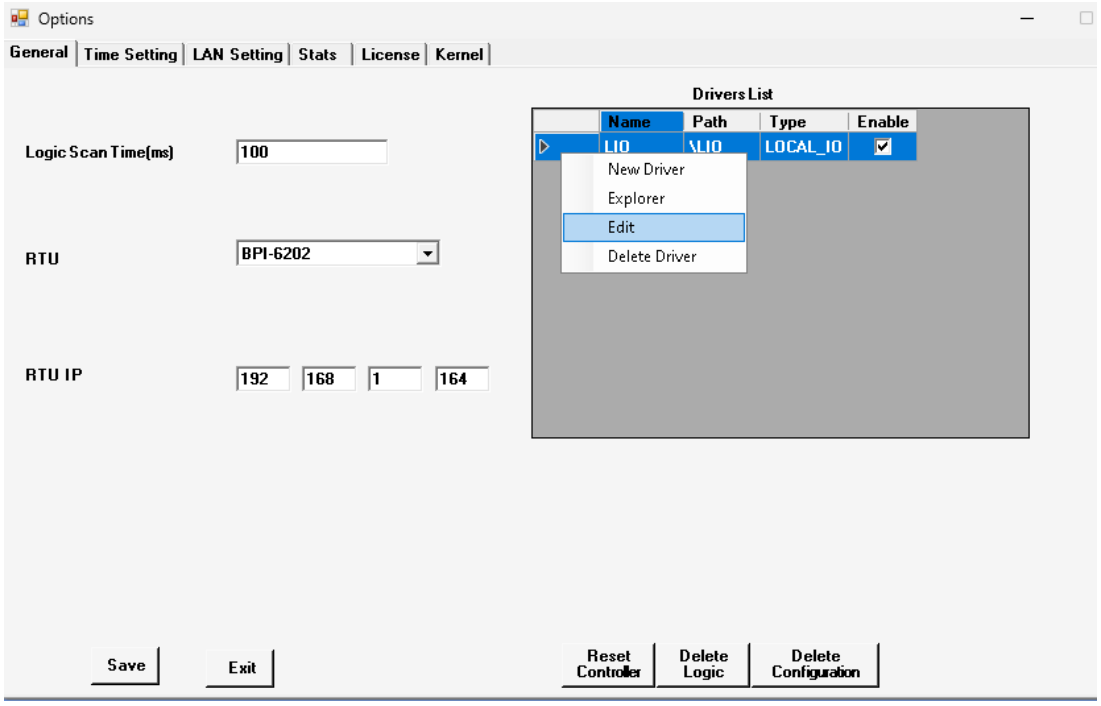


Click the "Make Driver" button. pbsSoftLogic will add the BPI-6202 configuration file to your project.

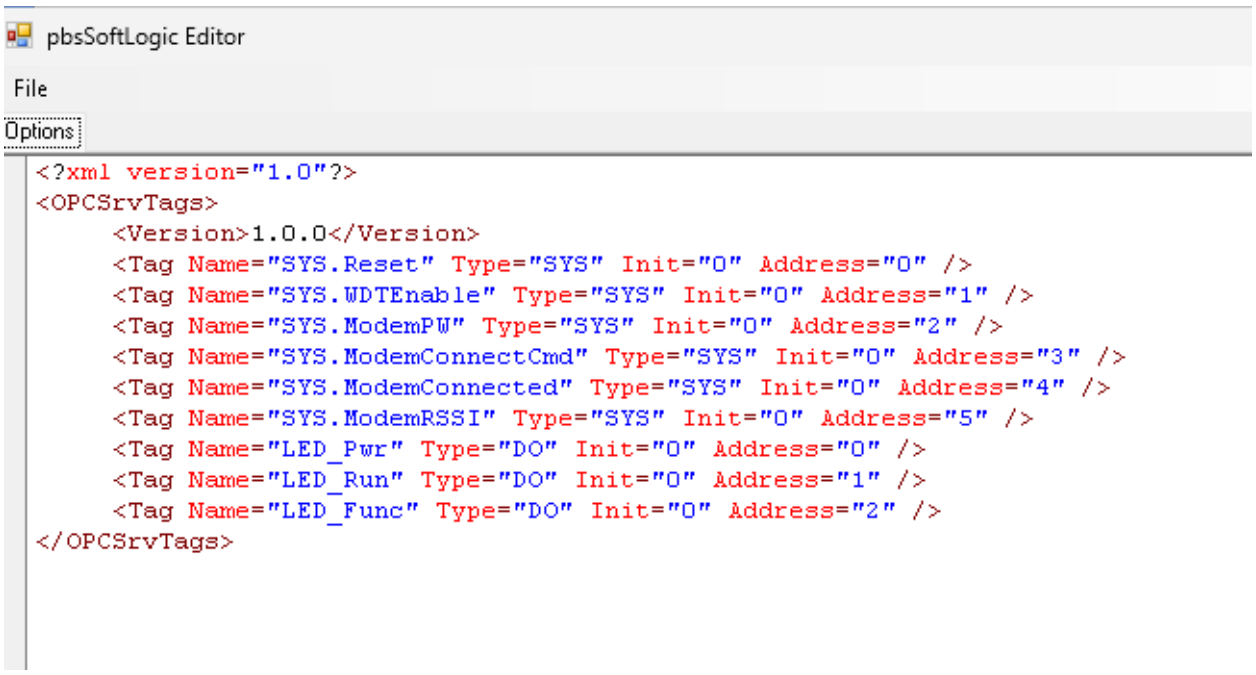


LOCAL_IO is a generic driver for modeling the hardware functionality of RTU. pbsSoftLogic creates different Local_IO configuration files based on the RTU type. Local IO manages all IO ports located on the main CPU, such as LED, WDT, Digital IO, etc.

Right click on LIO Driver and Select "Edit" option.



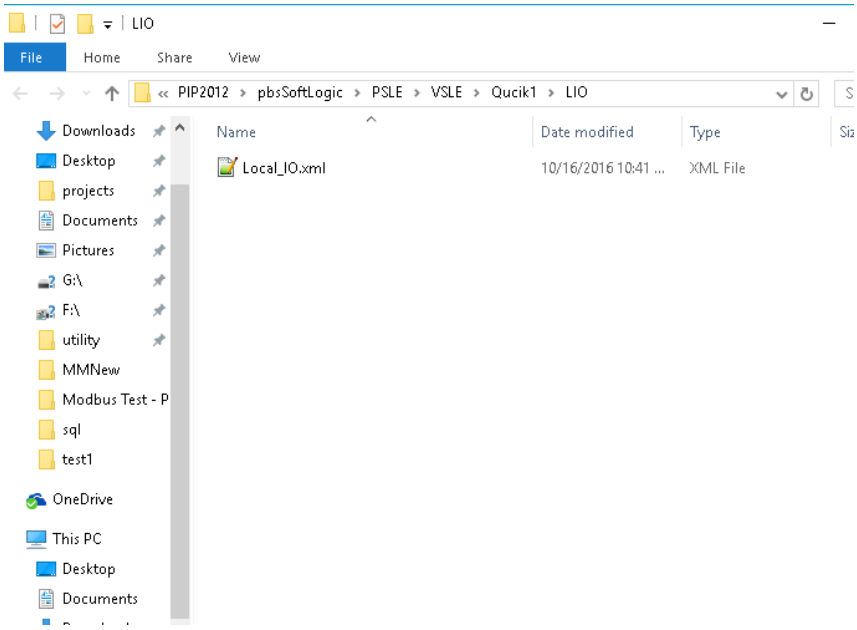
The default editor for the LocalIO Driver will open:




```
<?xml version="1.0"?>
<OPCSrvTags>
     <Version>1.0.0</Version>
     <Tag Name="SYS.Reset" Type="SYS" Init="0" Address="0" />
     <Tag Name="SYS.WDTEnable" Type="SYS" Init="0" Address="1" />
     <Tag Name="SYS.ModemPW" Type="SYS" Init="0" Address="2" />
     <Tag Name="SYS.ModemConnectCmd" Type="SYS" Init="0" Address="3" />
     <Tag Name="SYS.ModemConnected" Type="SYS" Init="0" Address="4" />
     <Tag Name="SYS.ModemRSSI" Type="SYS" Init="0" Address="5" />
     <Tag Name="LED_Pwr" Type="DO" Init="0" Address="0" />
     <Tag Name="LED_Run" Type="DO" Init="0" Address="1" />
     <Tag Name="LED_Func" Type="DO" Init="0" Address="2" />
</OPCSrvTags>
```

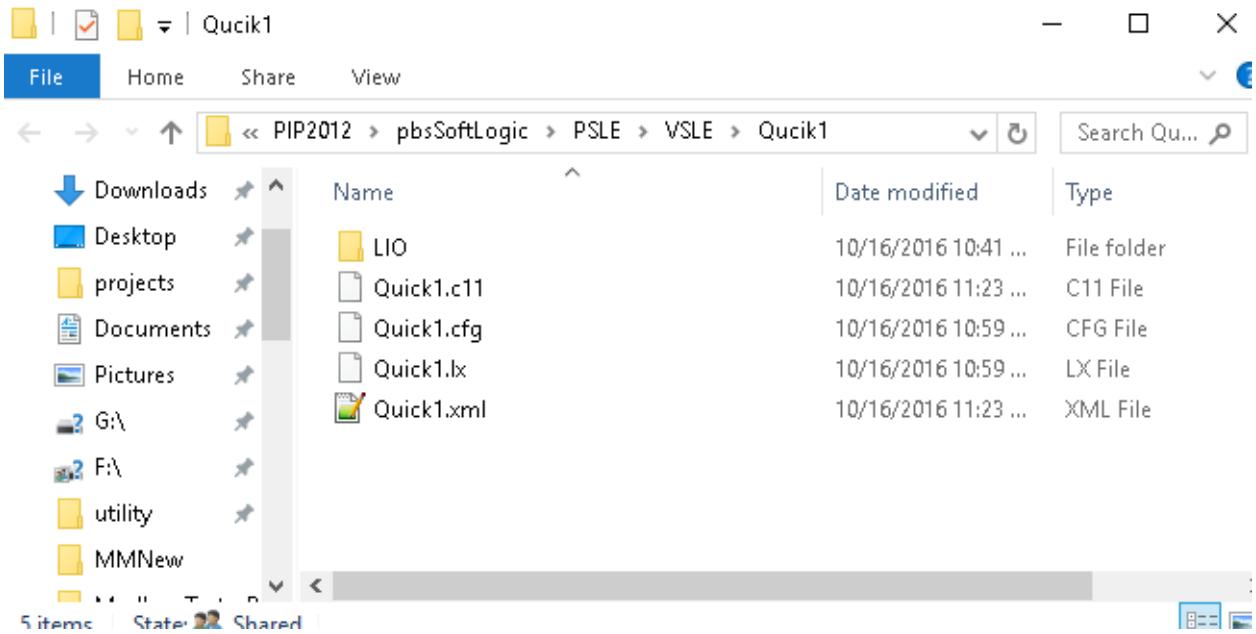
For more information about BPI-6202 local I/O, please refer to pbsSoftLogic Runtime Kernel Configuration for BPI-6202/04.

You can explore driver too, and open driver folder.



pbsSoftLogic creates a directory for each driver in the project directory.

If you open project directory you will see following file configuration:



Quick1.xml: source of your logic

Quick1.cfg: Project Configuration file.

Quick1.lx: Compiled Project Configuration file that is transferred to RTU

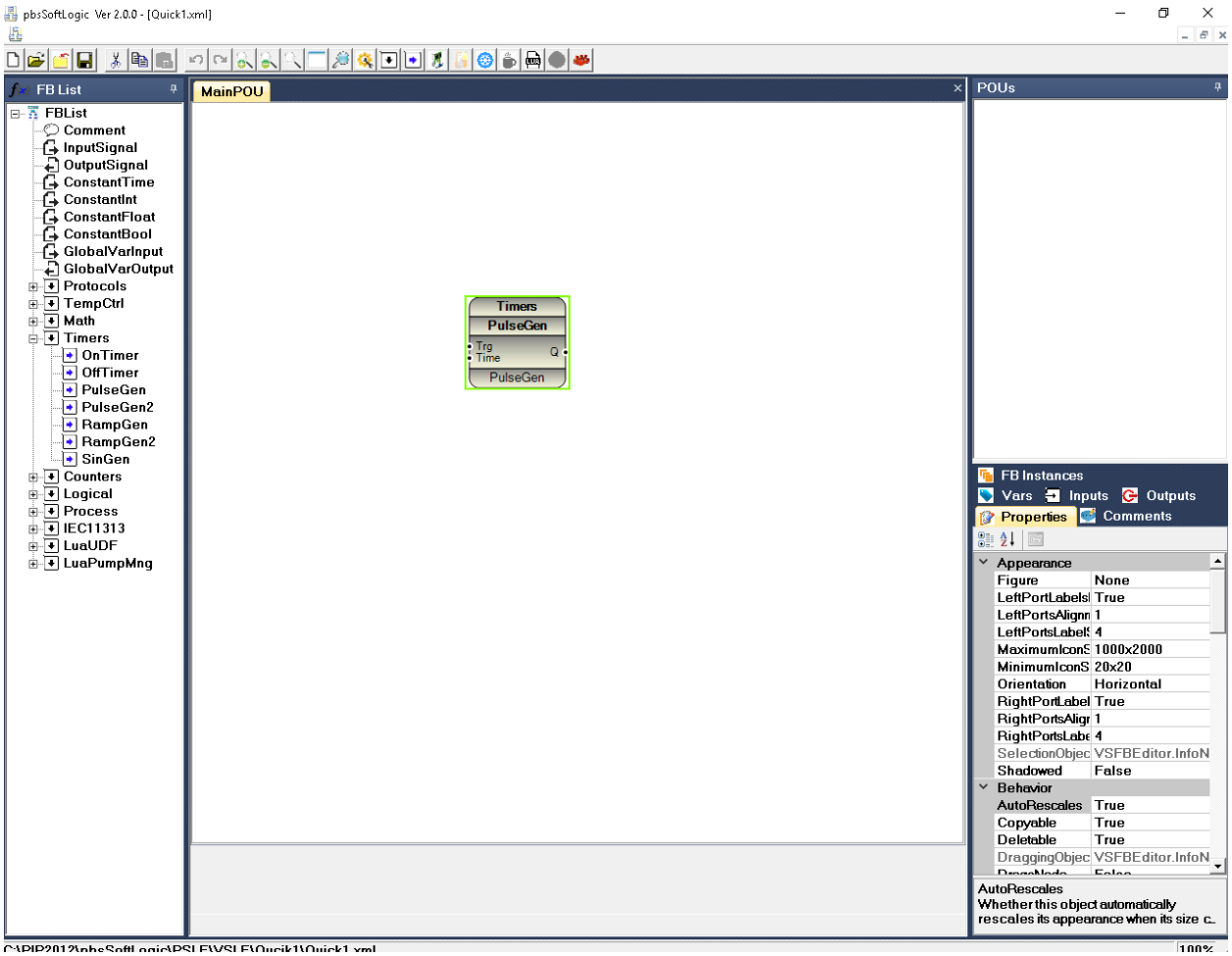
Quick1.c11: Compiled Logic that is transferred to RTU

LIO Directory: Directory for Local_IO Driver. Insider LIO directory, pbsSoftLogic is make Local_IO.xml file.

Step 4 Function Block Programming :

In the left panel, you can see various ready-made function blocks, and in the middle panel, the function block programming area.

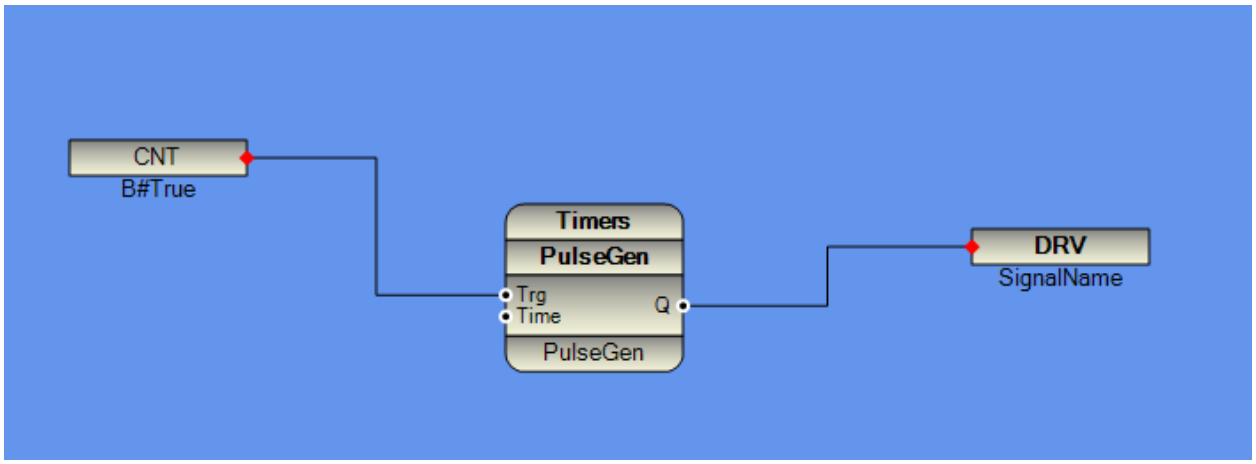
Open the Timers group and select PulseGen. Drag and drop it into the application area.



"PulseGen" generates continuous pulses with equal duration (low and high).

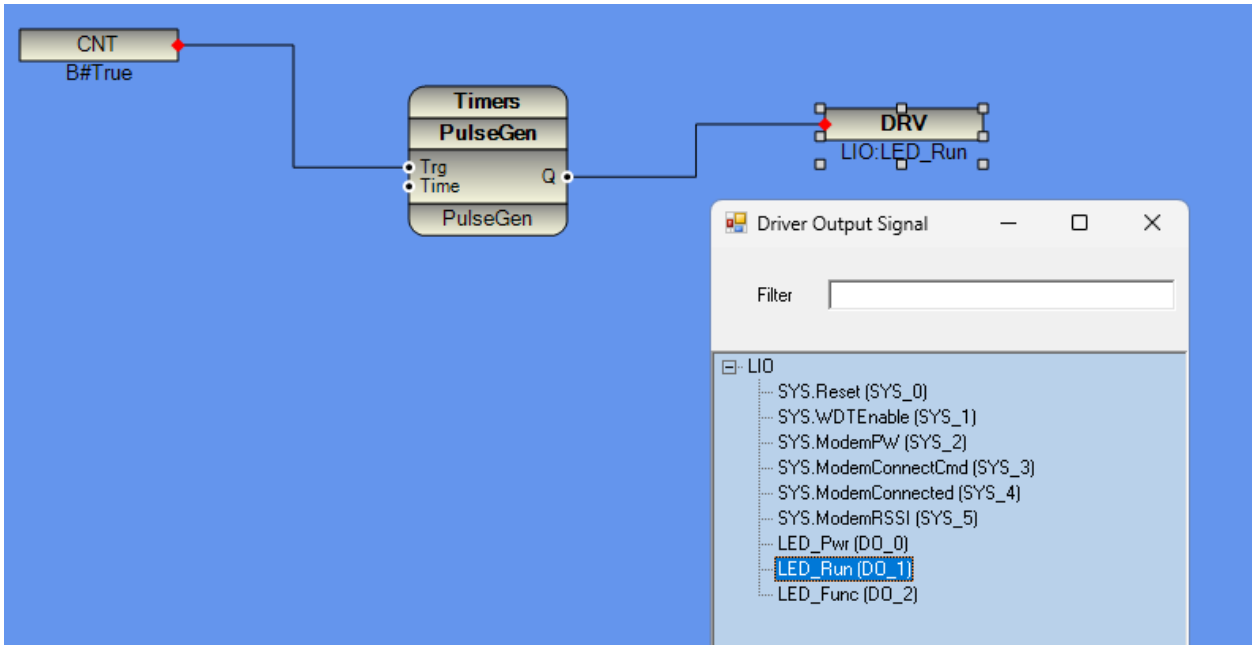
When the input "Trg" changes from low to high (0 to 1), the pulse train at the output "Q" starts with a low and high duration equal to the input "Time".

In the FB list panel, drag and drop the "ConstantBool" object and connect it to the "Trg" input. Then drag and drop the "OutputSignal" object and connect it to the "Q" output. Leave the "Time" input unconnected.



When an input port is not connected to any signal, it takes the default value that is preset for each input port (you can change the default input values of the input port).

Right-click on the output signal connected to the "Q" port and select the "LIO:LED_Run" signal.



Double-click on the "LED_Run" signal and close the "Driver Output Signal" form.

Click on Save and compile button at top.

It will save and compile your logic.

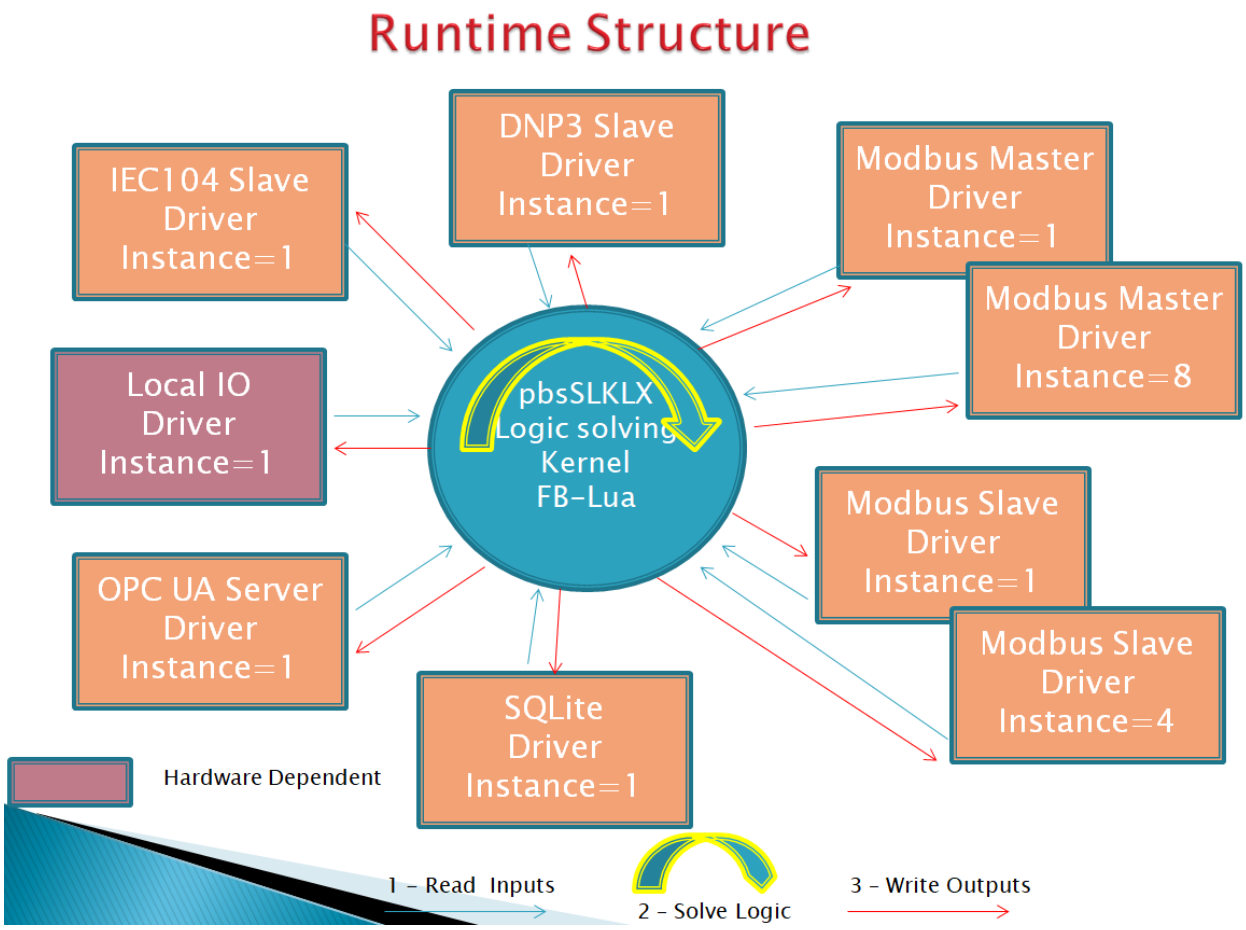
6– Runtime Kernel for Linux, transferring License to

In this chapter we will discuss the pbsSoftLogic runtime structure inside the RTU. The pbsSoftLogic runtime core is based on a very simple concept.

pbsSLKLX: This is the main program in Linux that starts automatically when the RTU boots or you can load it manually for diagnostic purposes.

Communication drivers: Various communication drivers supported by pbsSoftLogic.

In the figure below you can see how the above components are connected to each other.

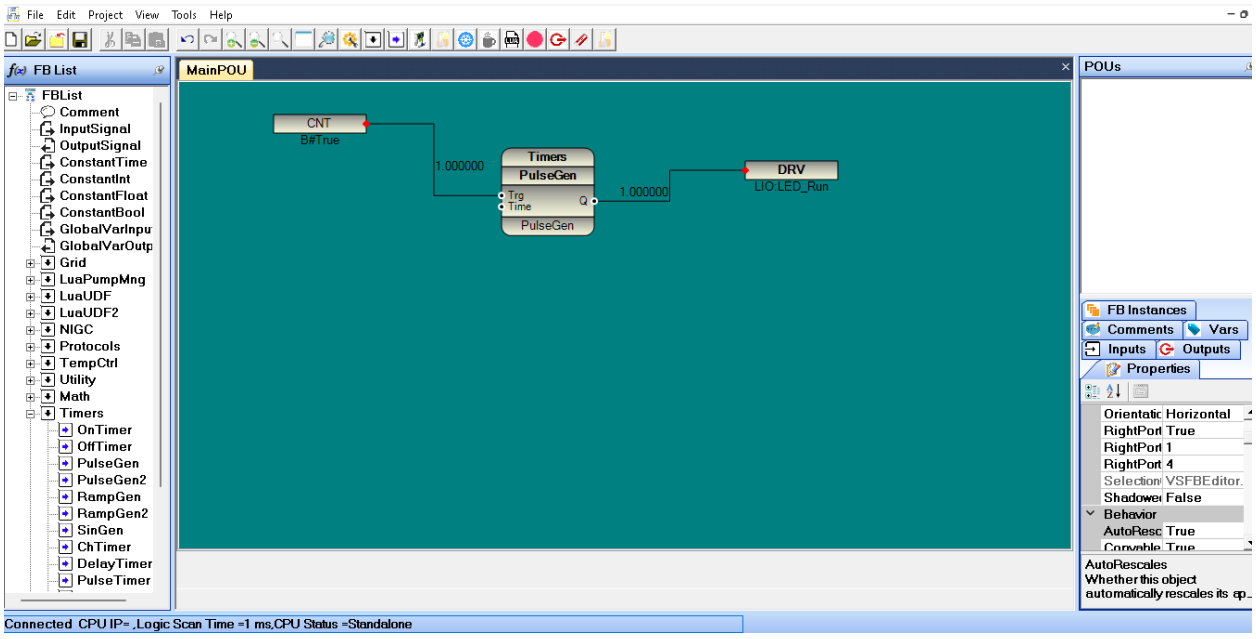


When you run pbsSLKLX, the following steps are performed on the RTU:

1. pbsSLKLX loads the file /home/pbsLX/logic.cfg. This file contains all the communication protocols defined for the active project.
2. pbsSLKLX dynamically loads the communication driver library into memory.
3. pbsSLKLX gets the driver parameters from the logic.cfg file and passes them one by one to the communication driver library.
4. pbsSLKLX adds the communication blocks, slots and finally the driver tags.
5. pbsSLKLX calls the pbsInit function to initialize the communication driver.
6. pbsSLKLX repeats steps 2, 3, 4, 5 for all the drivers defined in logic.cfg.
7. pbsSLKLX starts reading input tags for all communication drivers by calling pbsReadTag.
8. pbsSLKLX solves the RTU logic once.
9. pbsSLKLX writes all output tags of the communication drivers by calling the pbsWrite method.
10. Repeats steps 7, 8, 9.

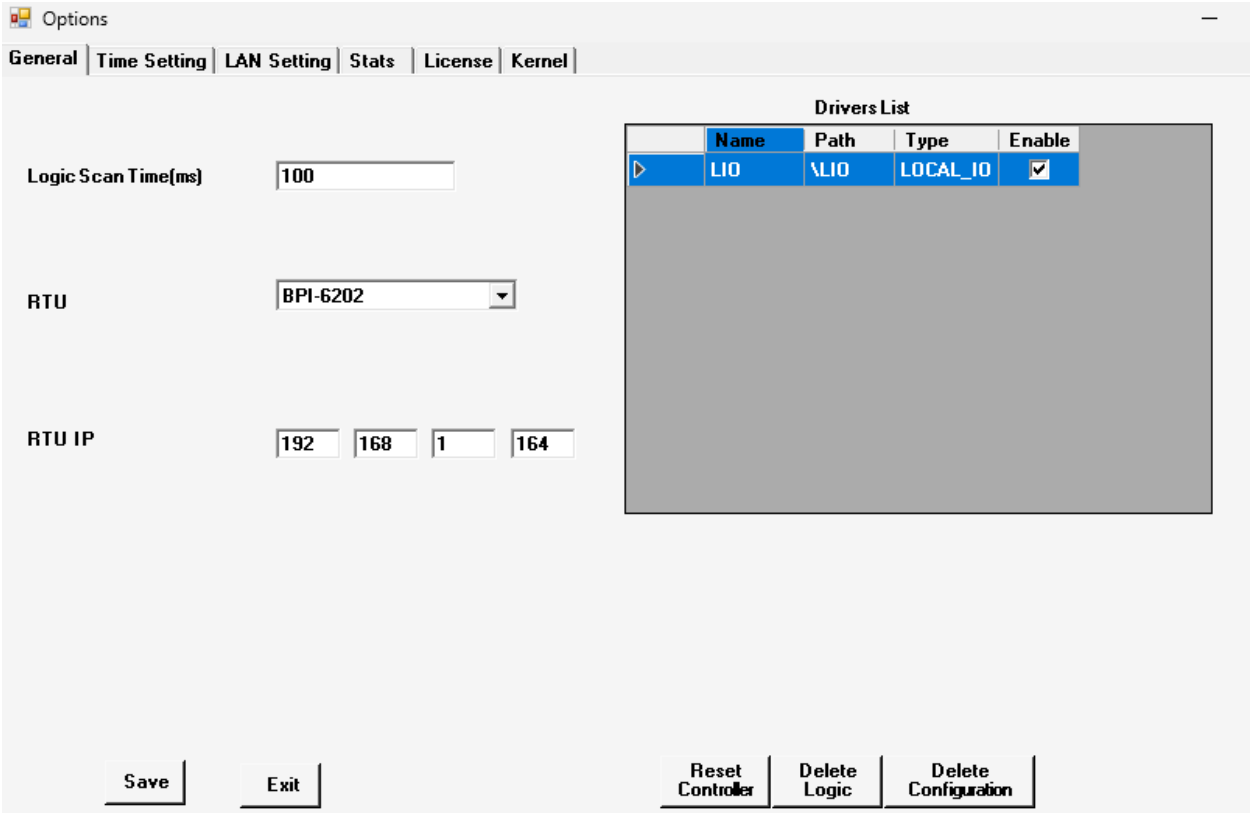
As is clear from the sequence above, the communication drivers have an integrated API interface to communicate with pbsSLKLX.

The execution time of steps 7, 8 and 9 is the actual logical scan time of the RTU. You can see the actual logical scan time when connecting to the RTU by pbsSoftLogic IDE. The actual logical scan time is shown at the bottom left of the screen. In the figure below, the actual logical scan time is 1 ms.



You cannot control the actual logical scan time, but you can set the logical scan time in the project settings page.

For this project logical scan time is 100 msec .



This means that every 100 milliseconds, the RTU reads all inputs defined in the drivers (step 7), executes the logic (step 8), and writes all outputs defined in the drivers (step 9). The execution time for steps 7, 8, and 9 is 1 millisecond. Therefore, the RTU processor is in sleep mode for 99 milliseconds.

pbsSLKLX has only one thread to read the driver input tags, execute the RTU logic, and write the driver output tags. But there is no limit in the driver library to take up a large number of CPU threads.

When you reduce the logical scan time and bring it closer to the actual logical scan time, the RTU processor utilization increases. Therefore, you should choose the logical scan time based on your process conditions and it should always be longer than the actual logical scan time.

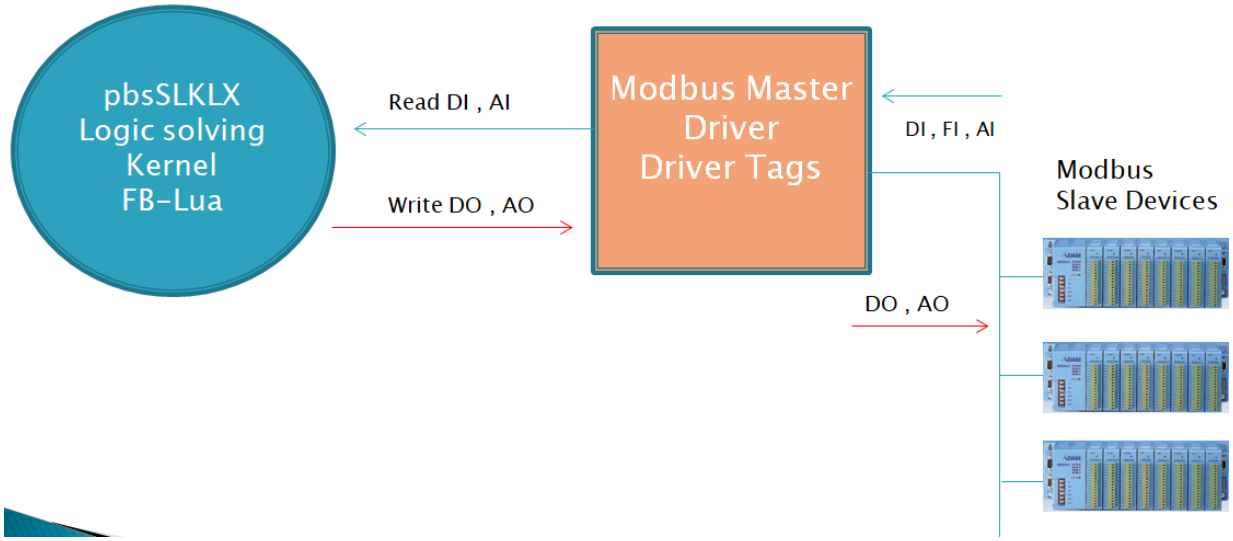
You can use less than 1ms for the logical scan time, setting 0.1 or 0.01ms is also possible.

Please note that the pbsSLKLX scan time is different from the communication protocol scan time. You can read Modbus Slaves every second, but for example, pbsSLKLX reads/writes driver tags every 10ms.

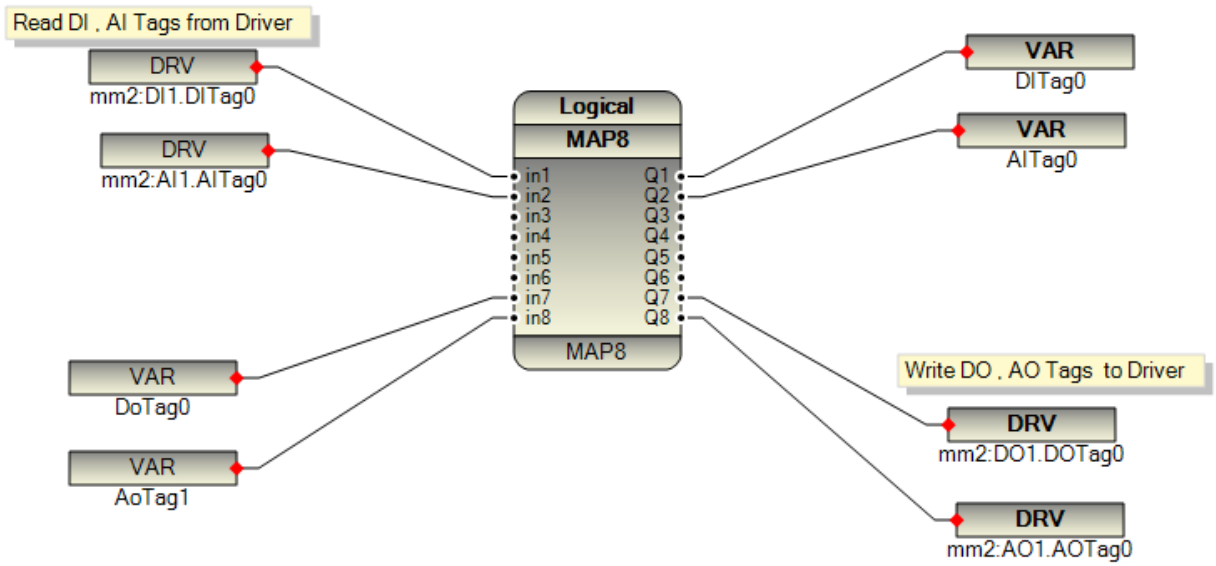
Slave and Master Drivers: There is a conceptual difference between Master drivers such as Modbus TCP/RTU Master, IEC101/104 Master and Slave drivers such as Modbus TCP/RTU Slave, DNP3 Slave, IEC101/104 Slave.

Master Drivers: In the figure below, you can see that the Modbus Master Driver reads digital and analog inputs from Slave devices and writes digital and analog outputs to Slave devices.

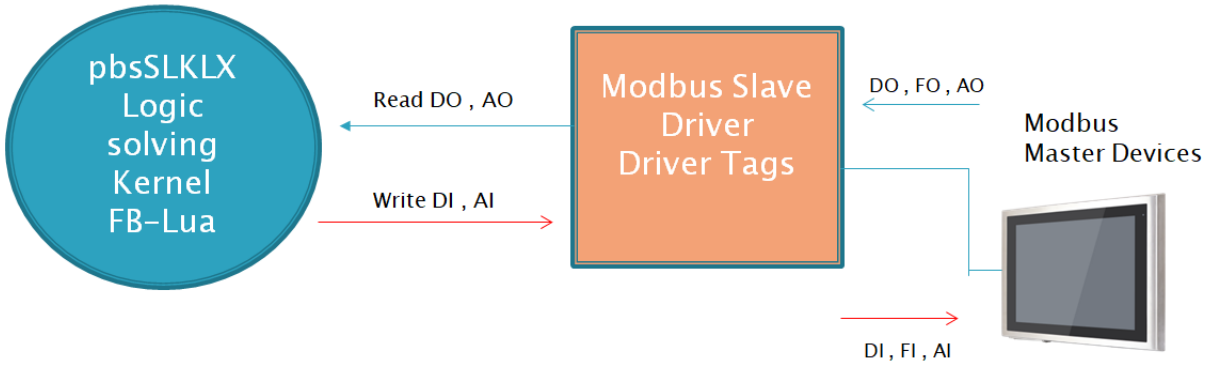
pbsSLKLX reads input digital and analog tags from drivers and writes output digital and analog tags to drivers.



In RTU logic, you must use the input signals on the left side of the function blocks to read from the driver, and you must use the output signals on the right side of the function blocks to write to the driver.

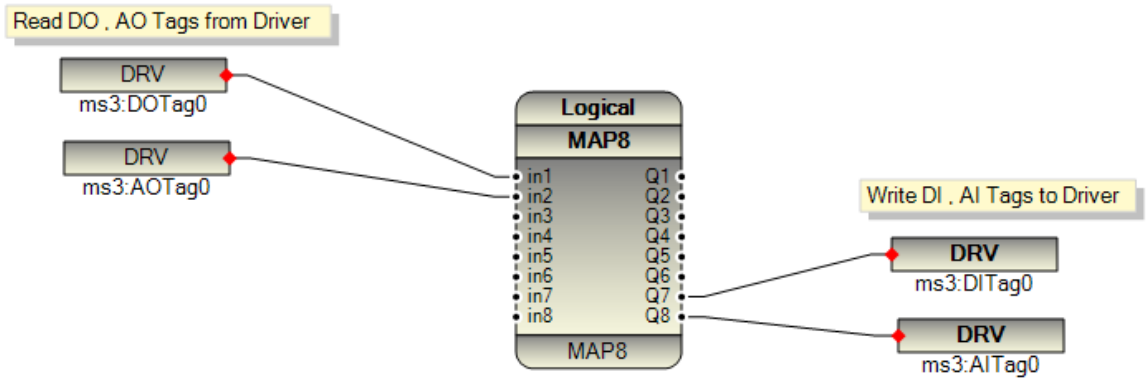


Slave Drivers: In the figure below, you can see that Modbus Master devices (HMI panels, SCADA software) read digital and analog tags from the Modbus Slave driver and write digital and analog output tags to the Modbus Slave driver.



pbsSLKLX writes digital and analog input tags to the Modbus Slave driver and reads digital and analog outputs from the Modbus Slave driver. These tags are written to the Modbus Slave driver by the external Modbus Master device.

In slave drivers, you should use DO and AO tags that are read from the driver on the left side of the function blocks and DI and AI tags that are written to the driver on the right side of the function block.



Linux Run Time Kernel

pbsSoftLogic has two parts:

1 - Engineering Station. Runs on Windows operating system

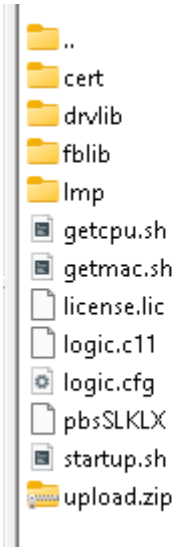
2 - Runtime Engine. Runs on Linux embedded inside RTU

In this part we will talk about Linux runtime engine.

The pbsSoftLogic runtime engine for Linux has the following modules:

- It locates at /home/pbsLX directory
- /home/pbsLX/pbsSLKLX file is main runtime module. It is an executable Linux file.
- /home/pbsLX/lmp/ logic monitoring protocol, for monitoring logic by IDE and Lua Interface module
- /home/pbsLX/fblib/ pbsSoftlogic internal, user and lua function blocks. For each FB group there is one dynamic library.
- /home/pbsLX/drvlib/ driver folder
- The "License.lic" file is for the runtime license. Without the license file, the program will run for 30 minutes.
- "Upload.zip" - When you transfer a project to RTU, the last project folder is always transferred as the "upload.zip" file. You can use this file in case you lose the project.
- "getcpu.sh" - Used to read CPU usage by LocalIO.
- "getmac.sh" - Generated by the runtime kernel to check license.

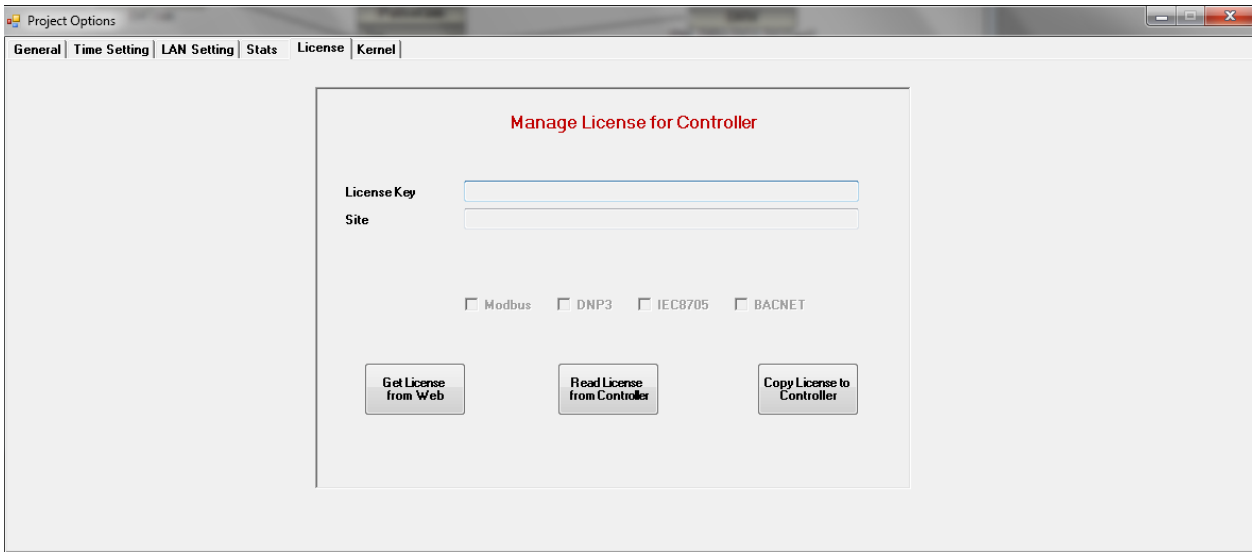
When you unzip the kernel runtime, you can see the following directories:



For each RTU, you need a license file for normal operation. Without a license, the device will run for 30 minutes and you will need to restart the controller.

When you purchase a pbsSoftLogic license, you will receive a license key. To activate the license, follow these steps:

1- Open the project settings page and select the License tab.



2 - Make sure that your computer on which pbsSoftLogic IDE is installed is connected to the Internet and the controller is also connected to the network with the computer.

3 - Copy the controller license key and paste it into the License key text box.

4 – Write a description of your project, project name, location, country,

5 – Click on "Get License from Web" . You will connect to the pbsControl website and get the licenses.

6 – Modbus , DNP3 , IEC and BACNet check boxs will be checked normally.

7 – Click "Copy License to Controller." This will transfer the license file to the controller.

8 – Restart Controller.

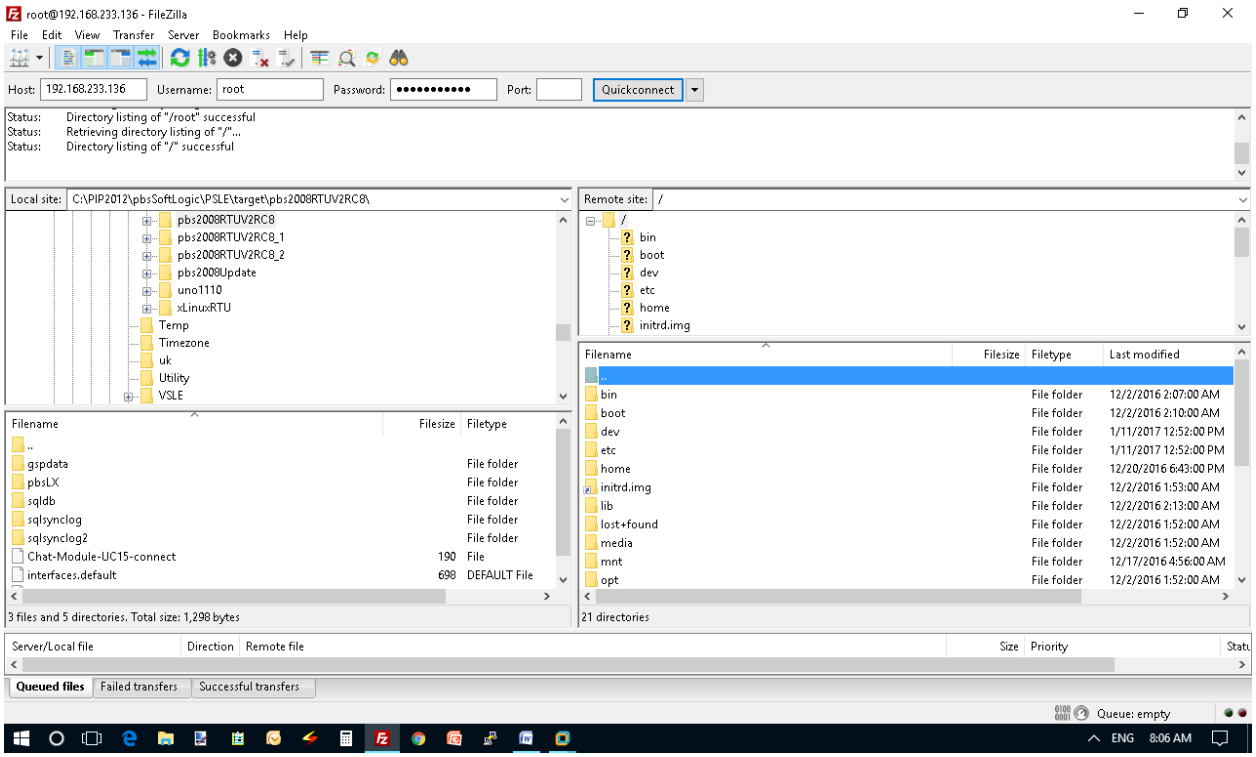
If you have a controller and want to check its license, click "Read license from controller."

Working with FileZilla

You can use FileZilla client utility to explore and edit RTU Files and directories.

Download filezilla from https://filezilla-project.org/

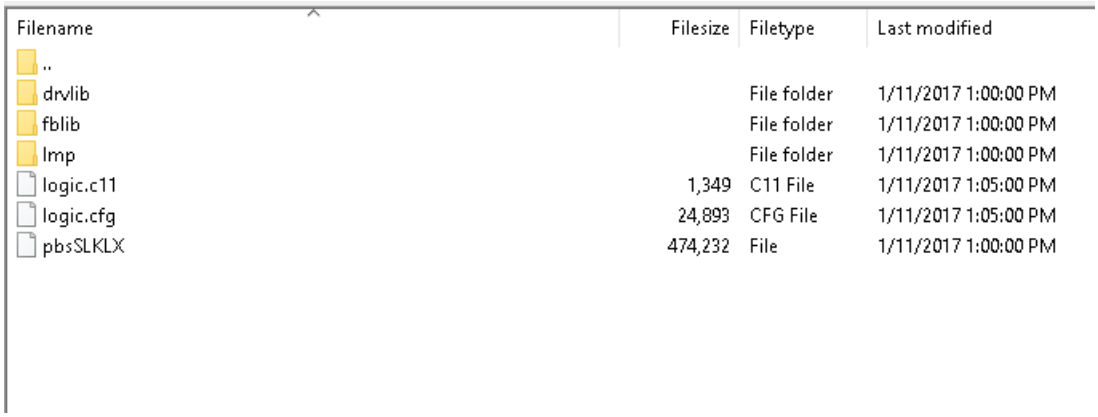
Run filezilla client you will see following page :



Type RTU IP at host field. Type root and root password in user name and password fields.

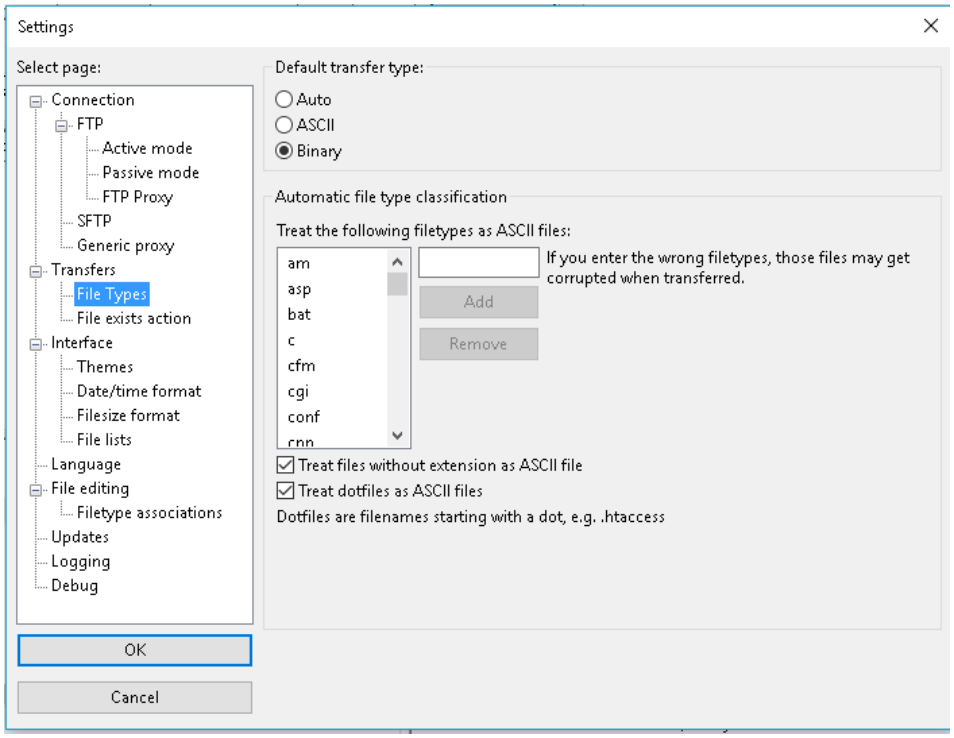
All RTU directories are showing at right panels and your PC directories at left s panels.

Double click on home directory. You will go inside home directory. Double click on pbsLX .pbsLX directory is runtime kernel of pbsSoftLogic for Linux Operating system .



If you install debian X86 Linux on your pc over VMWare or Virtual Box, then you can easily use pbsSoftLogic Debian X86 Runtime and test all functionalities of pbsSoftLogic on your PC.

Note : for transferring files between Windows and Linux Systems , always set Transfer File Type to Binary. you can find this option in Edit Menu , Setting menu and Transfers Segment .



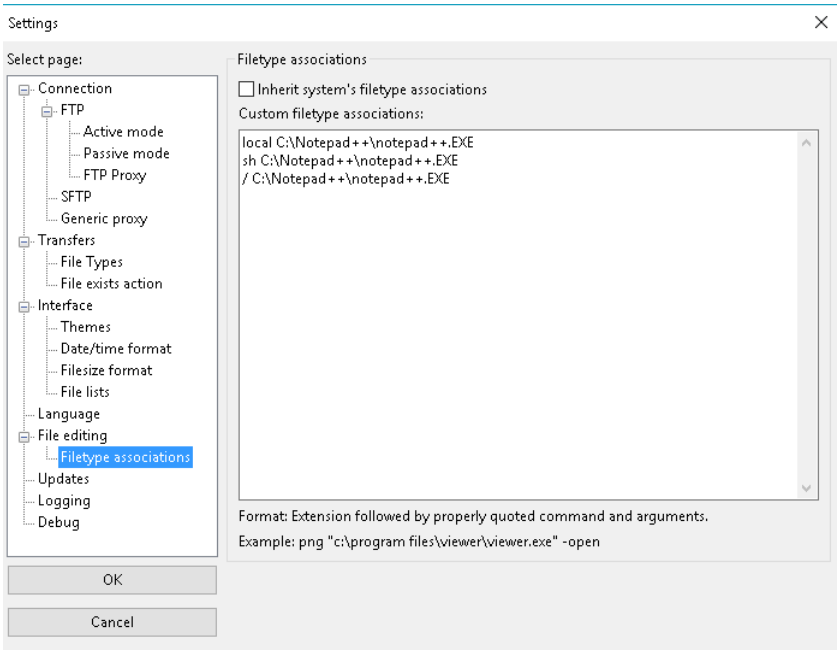
By default it is set to "Auto" that is damaging Linux files at transfer time from Windows to RTU.

For editing RTU configuration files in windows you need to use NotePad++ Editor to not damage Text file format when transfer to windows System .

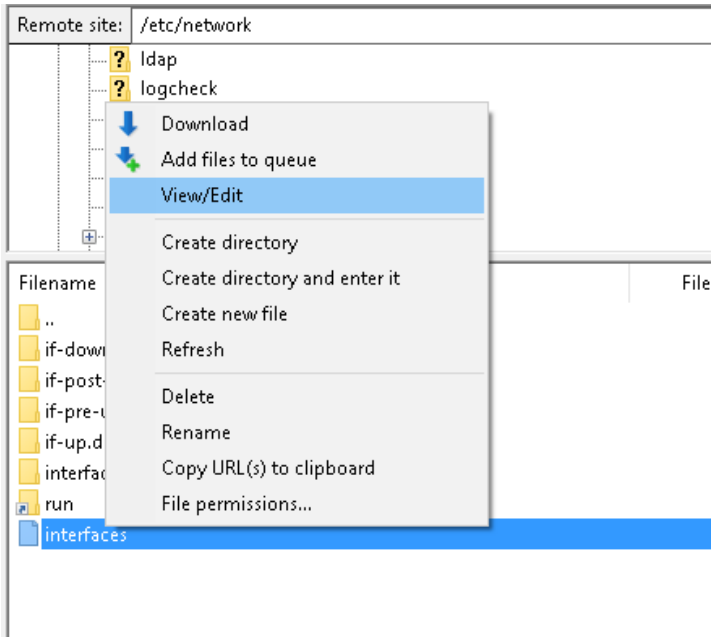
Install NotePad++ utility from https://notepad-plus-plus.org/

At first time that you View/Edit any Linux Configuration file , Filezilla will ask you for Custom Editor .

In this Stage set Nodepad++ as default editor in Filezilla . This will change File Editing Option in Setting page as following :



For changing Network Interfaces in RTU , View/Edit /etc/network/interfaces file .



For transferring files from RTU to Windows, Select File, Right click on name of File or Directory and run "Download" command.

You can delete or Rename files inside RTU by selecting file, right click on name of file and select Delete or Rename Commands.

7 – Project Settings facilities

There are many facilities in setting page in pbsSoftLogic Editor.

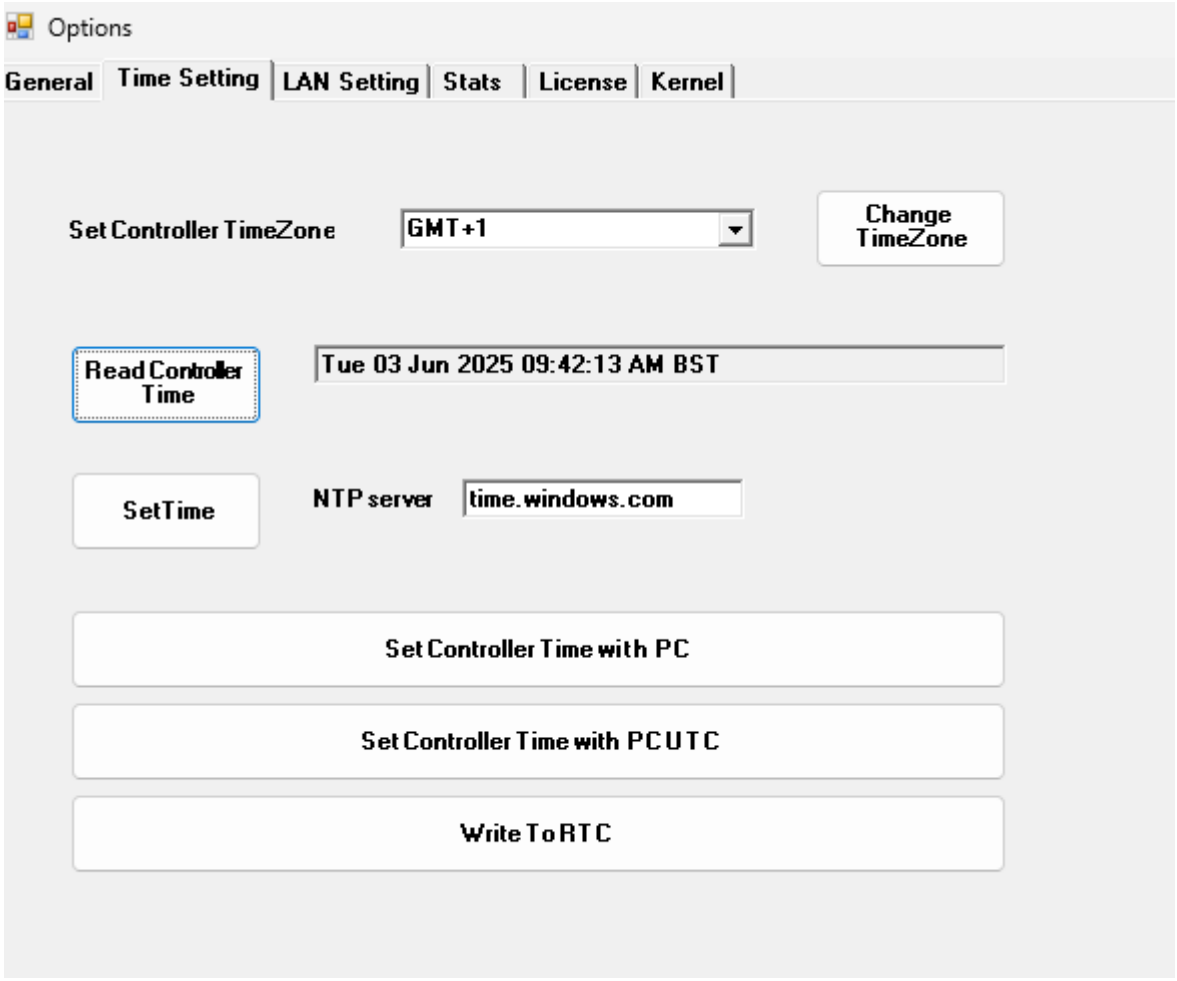
Open Setting Page you can see following tabs:

- General
- Time Setting
- LAN Setting
- Stats
- License
- Kernel

General Tab: In This page you can set following parameters:

- Logic Scan time (Msec)
- Controller Type
- Controller IP address
- Communication Drivers
- Restart Controller
- Delete Logic
- Delete Configuration

Time Setting:



Set the controller's time zone: Select your location from the list box and click Change time zone.

Read Controller Time: Reads the current date, time, and time zone of the controller.

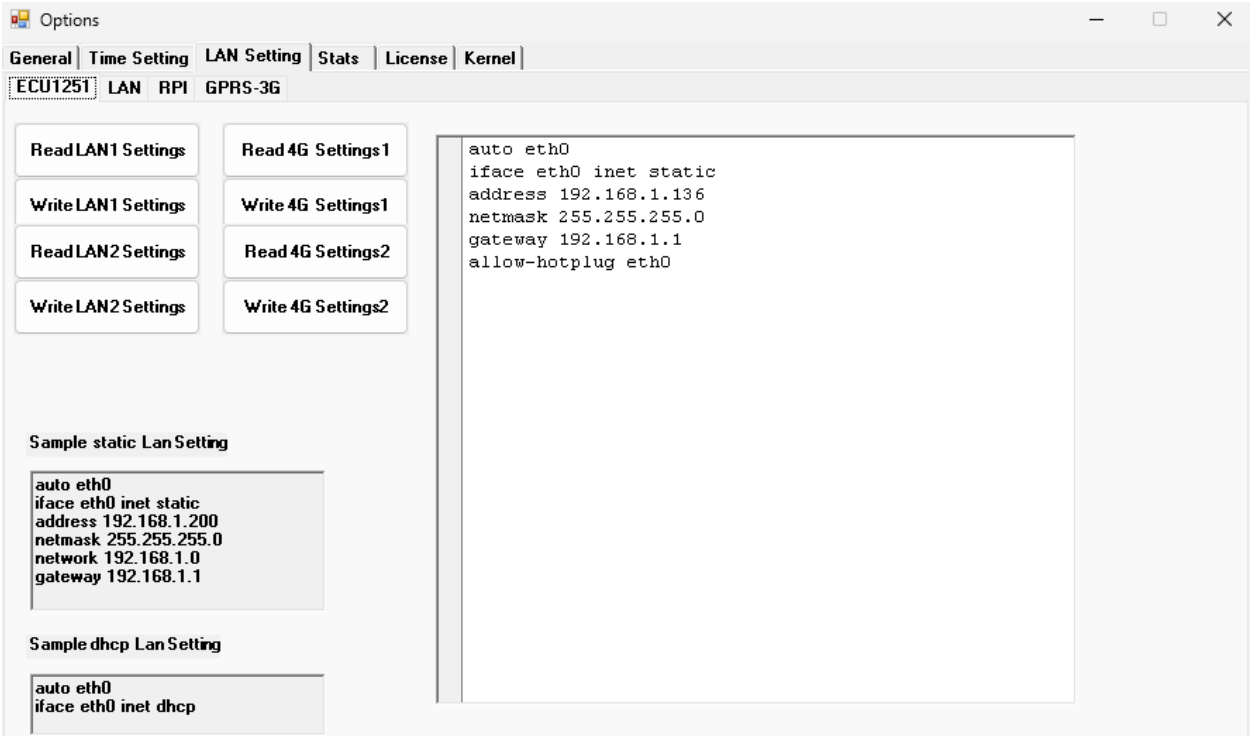
Time Setting: Sets the controller's time from an NTP server, which can be a computer on the network or any time website. However, the controller must be connected to the Internet.

Set controller time with computer: This command sets the controller time from the computer running pbsSoftLogic.

Set controller time with PC UTC: This command sets the controller time from the computer running pbsSoftLogic in UTC.

Write to RTC: Write the latest time to the controller's RTC.

LAN Settings:



Depending on the RTU type, you can use different pages to configure the LAN. For ECU1251D, you can see the following functions:

Read LAN1/2 Settings: Reads the current LAN1/2 settings from the controller.

Write LAN1/2 Settings: Writes the new LAN1/2 settings to the controller.

To change the controller IP address:

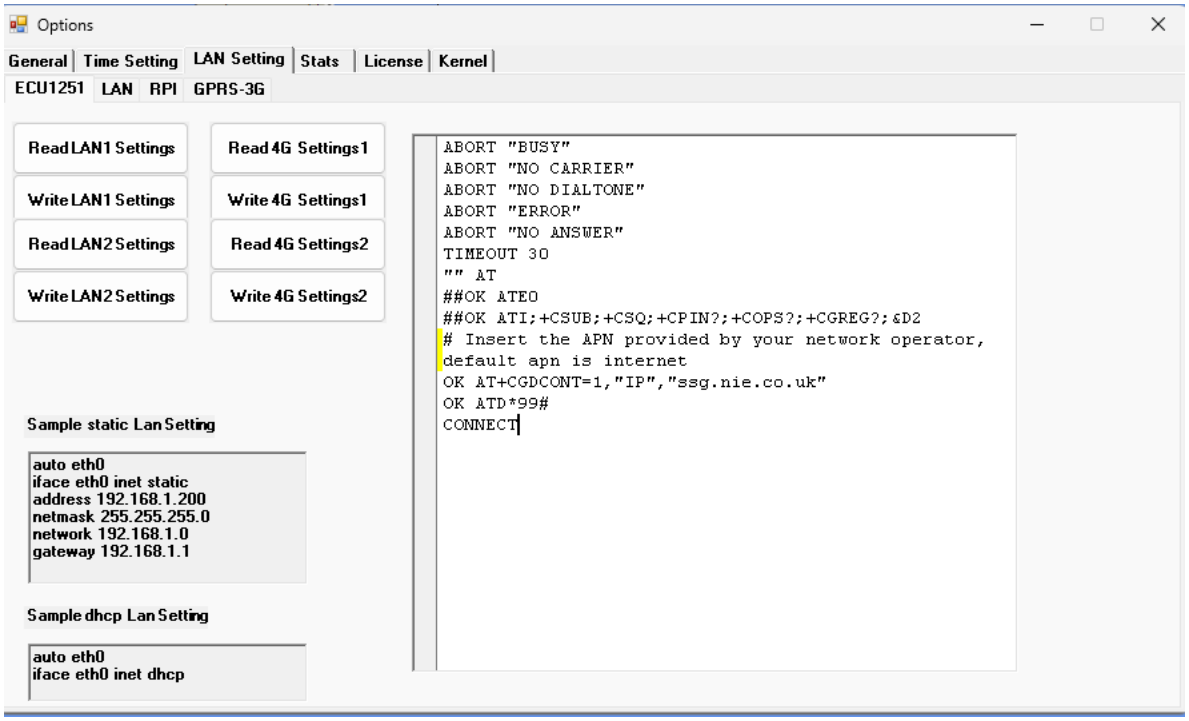
1 – First read the LAN settings.

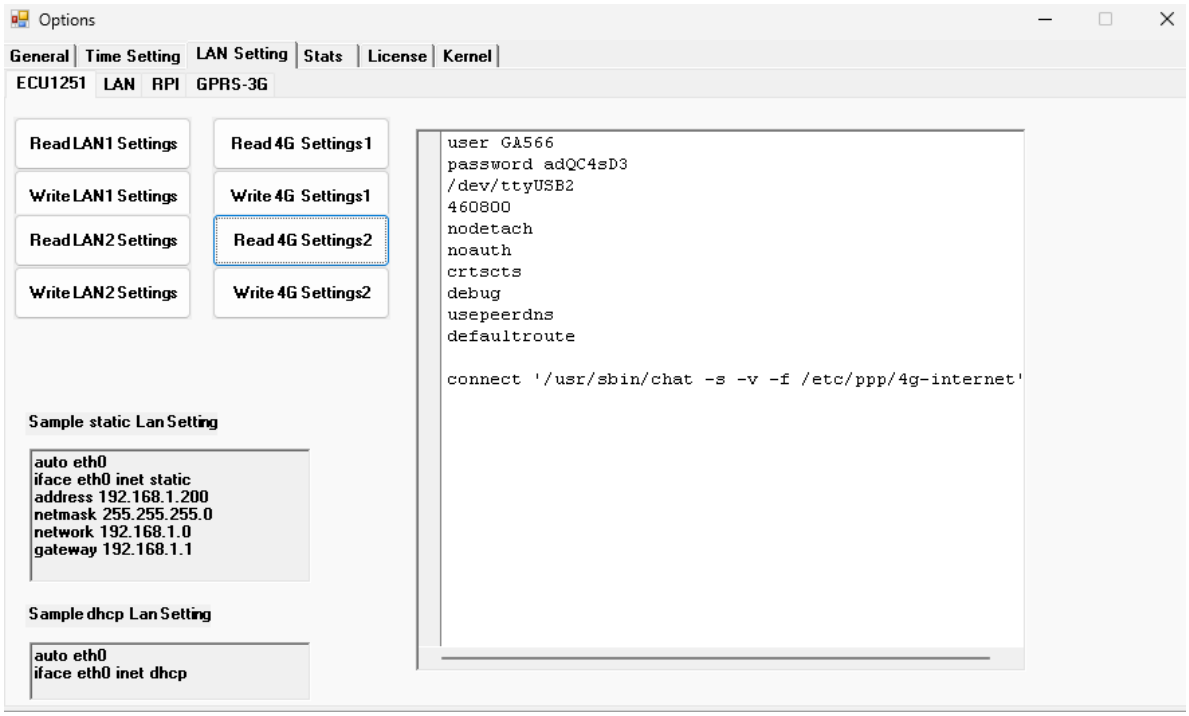
2 – Change the IP address for each LAN port and other settings.

3 – Write the new settings to the controller.

Setting 4G for ECU1251D

There are two files for 4G settings in ECU1251D. As with LAN settings, first read the configuration, make changes and write the new settings to RTU.

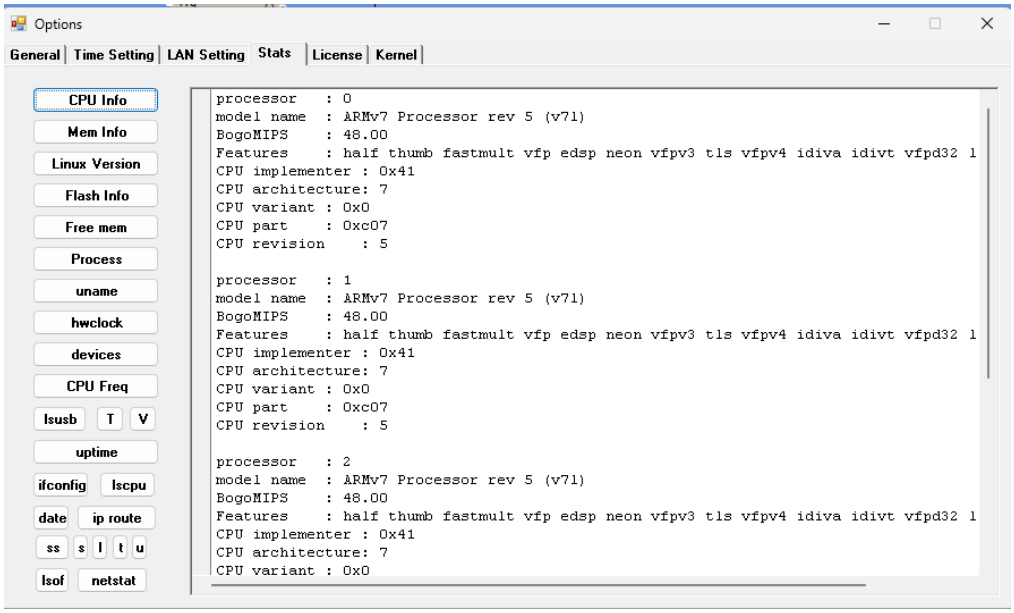




Controller Stat tab:

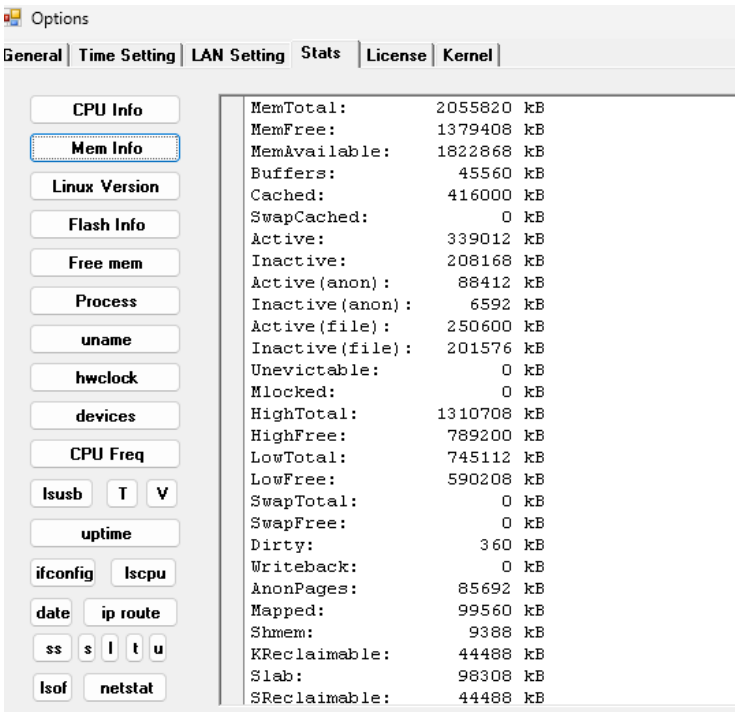
Read CPU Information: It will Read Hardware Information from controller

Linux Command: `cat /proc/cpuinfo`



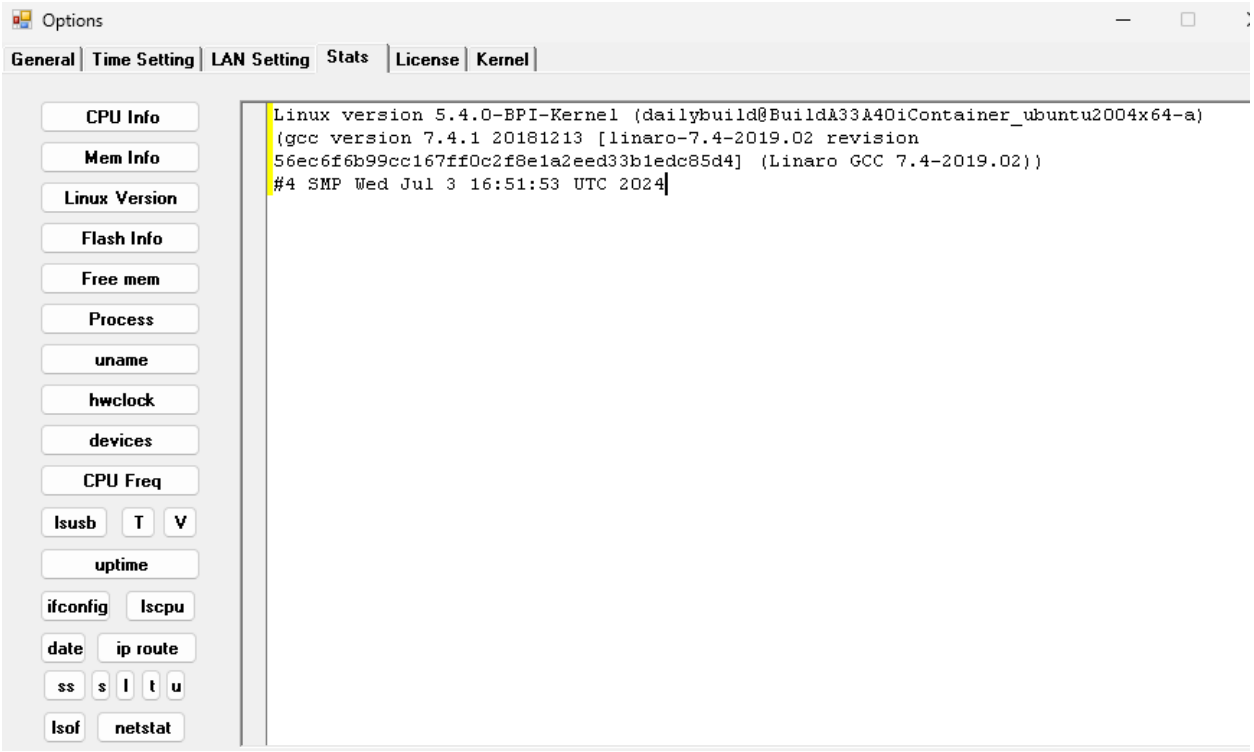
Read memory information: shows detail of memory usage of controller

Linux Command : `cat /proc/meminfo`

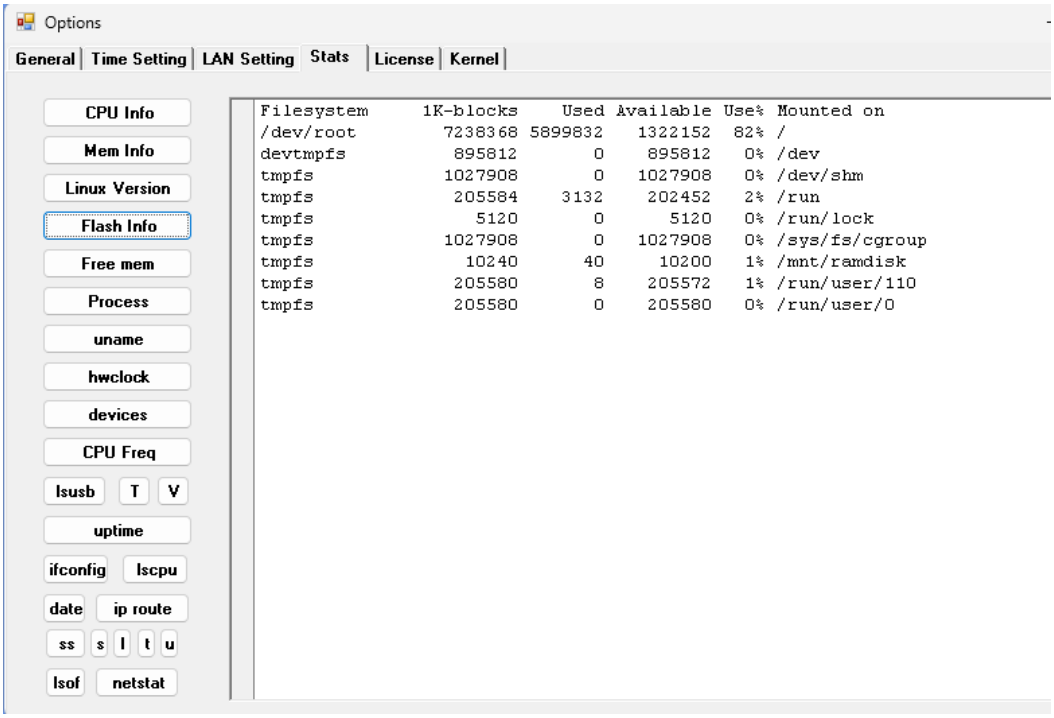


Read Version: Read Controller Linux Version, GCC compiler version

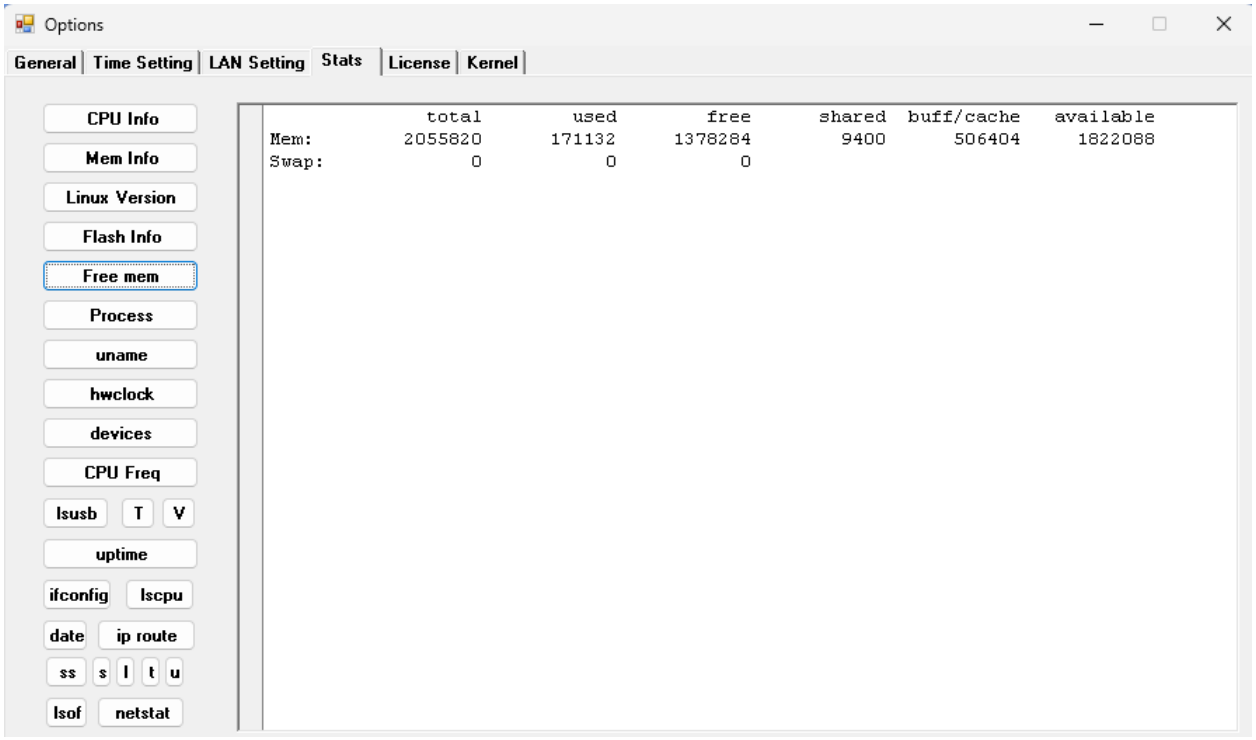
Linux Command : `cat /proc/version`



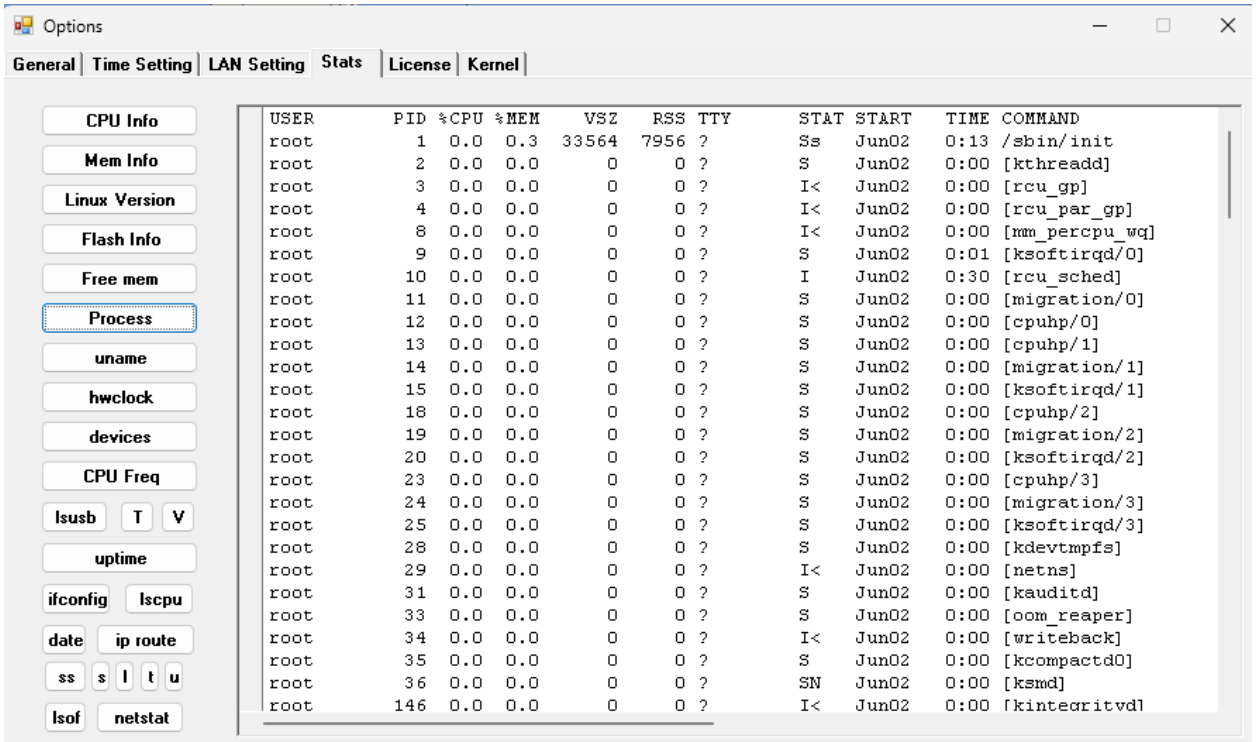
Read Flash Information: Linux Command : `df –h`



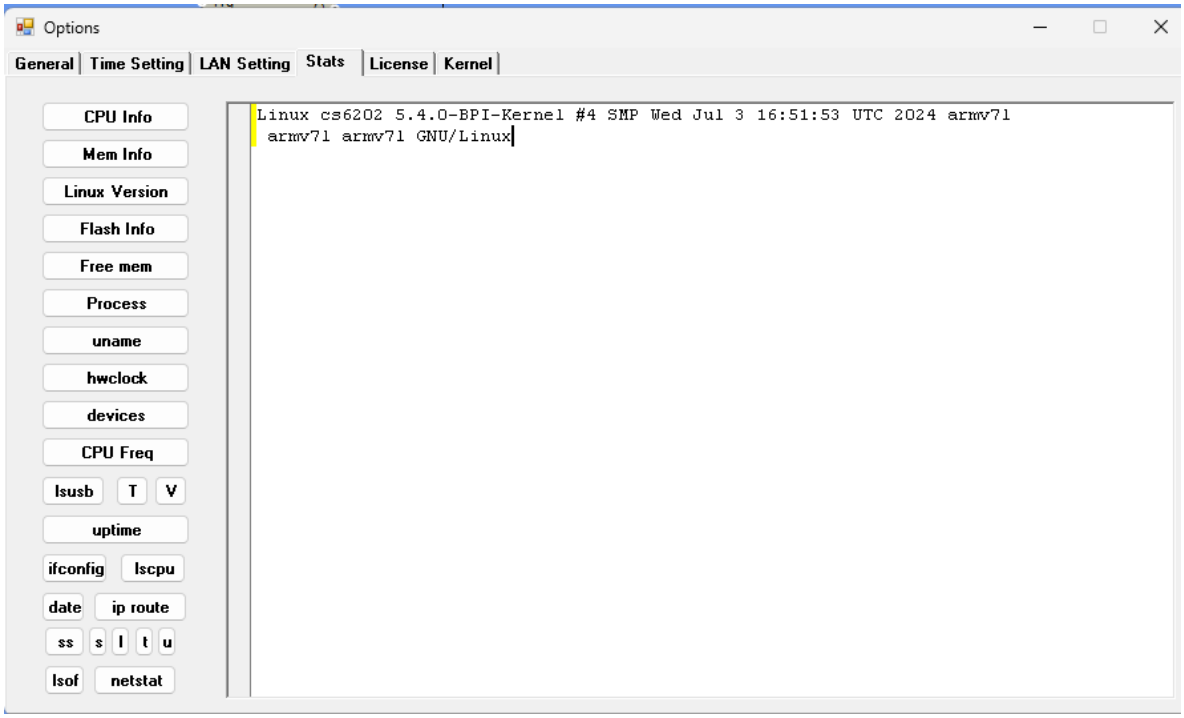
Read Free Memory: Linux Comamnd `free`



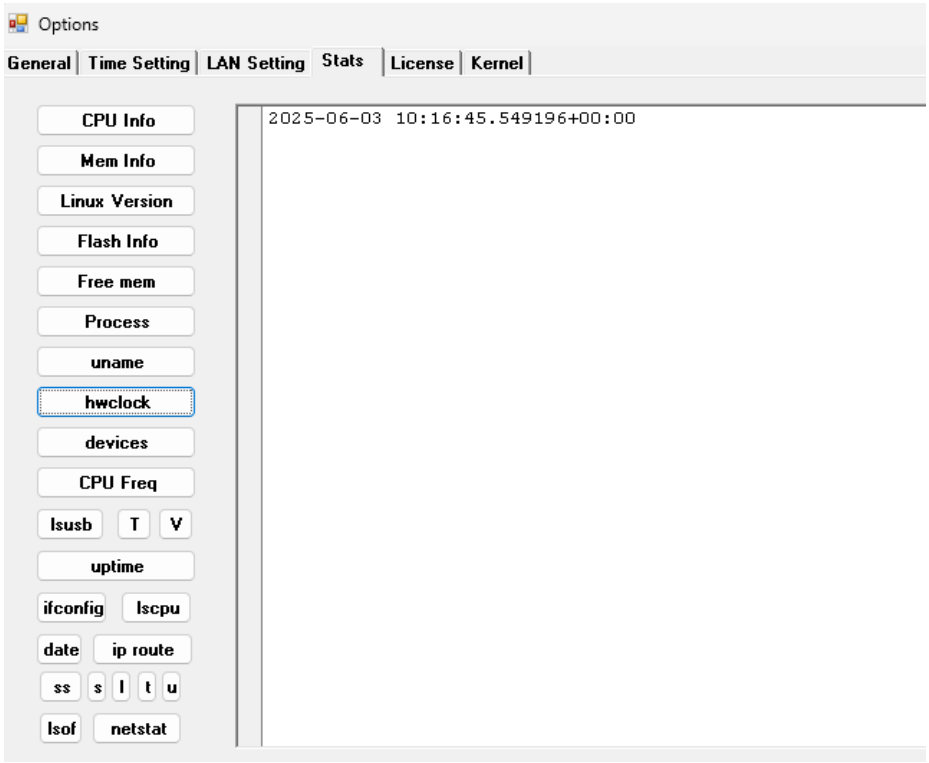
List of Process : Linux Command : `ps aux`



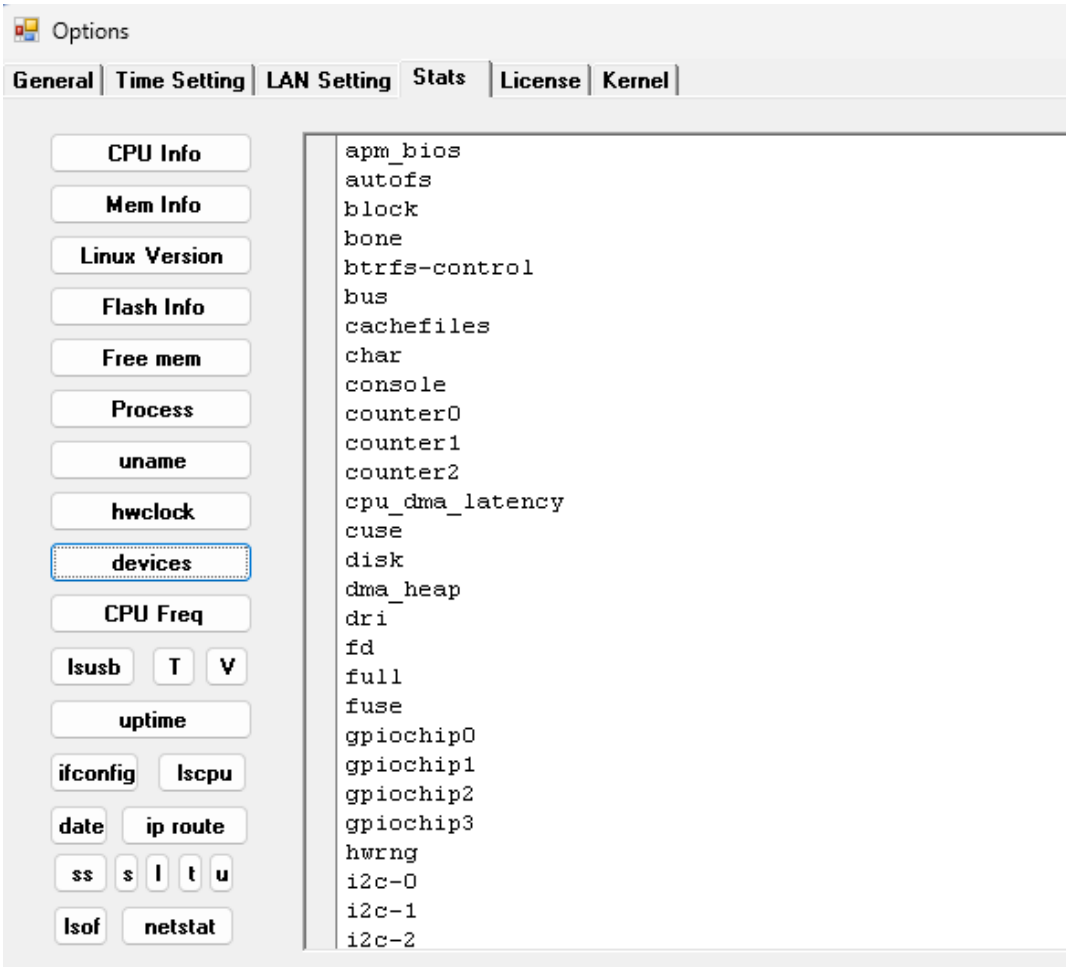
uname : linux command : `uname -a`



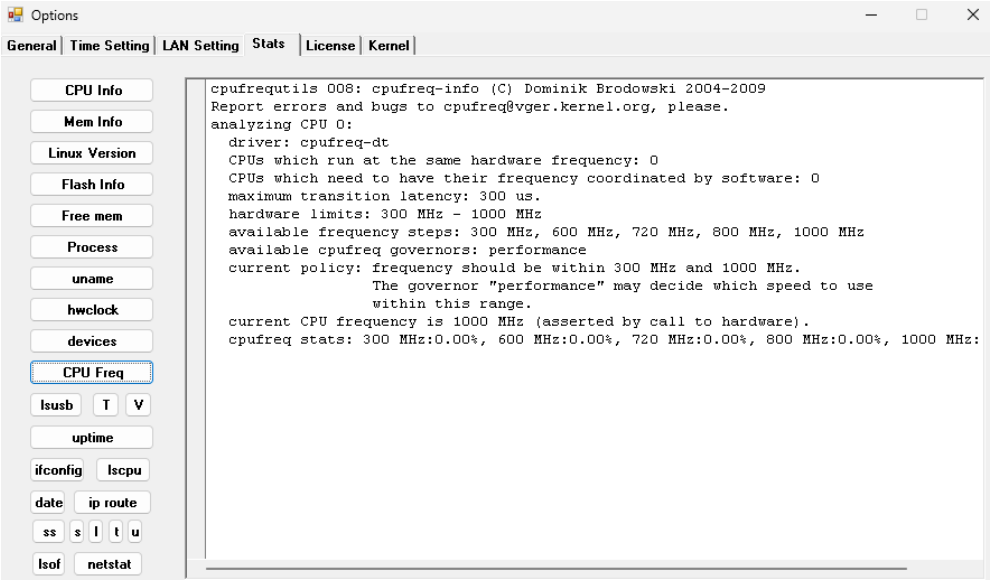
hwclock : Linux command : `hwclock`



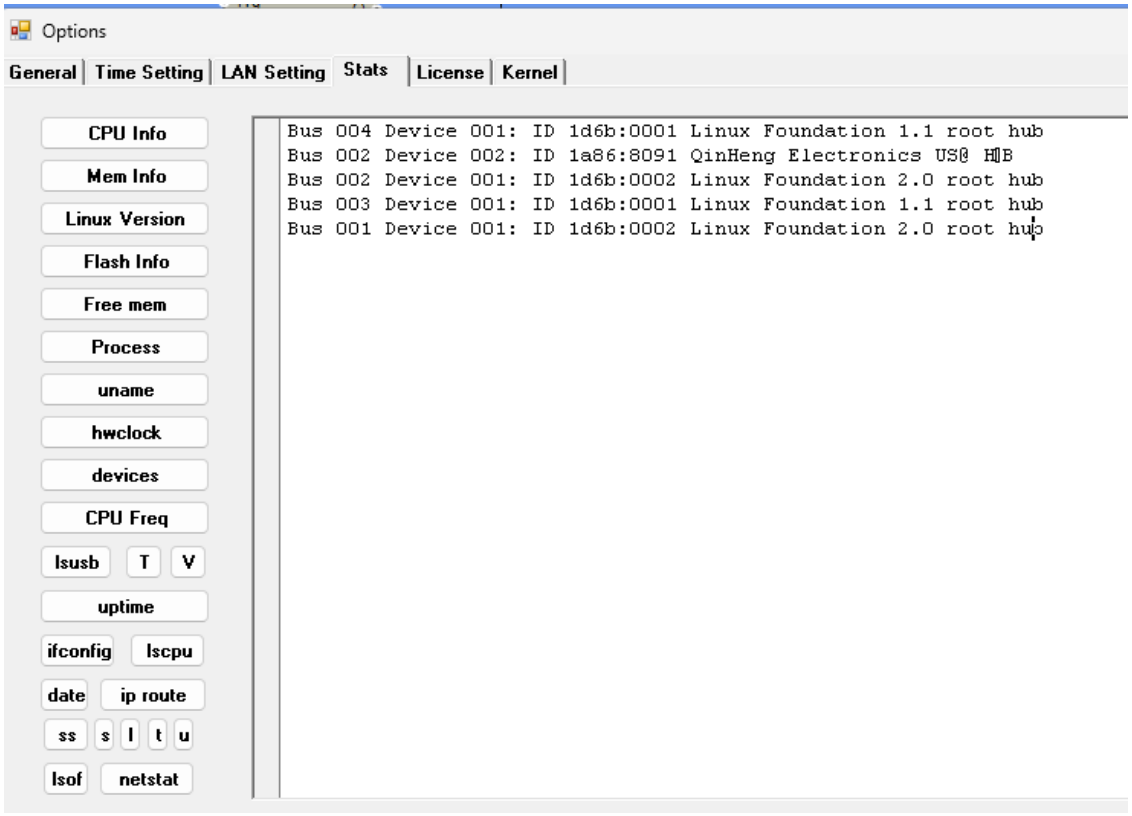
Devices : Linux command : `ls /dev/`



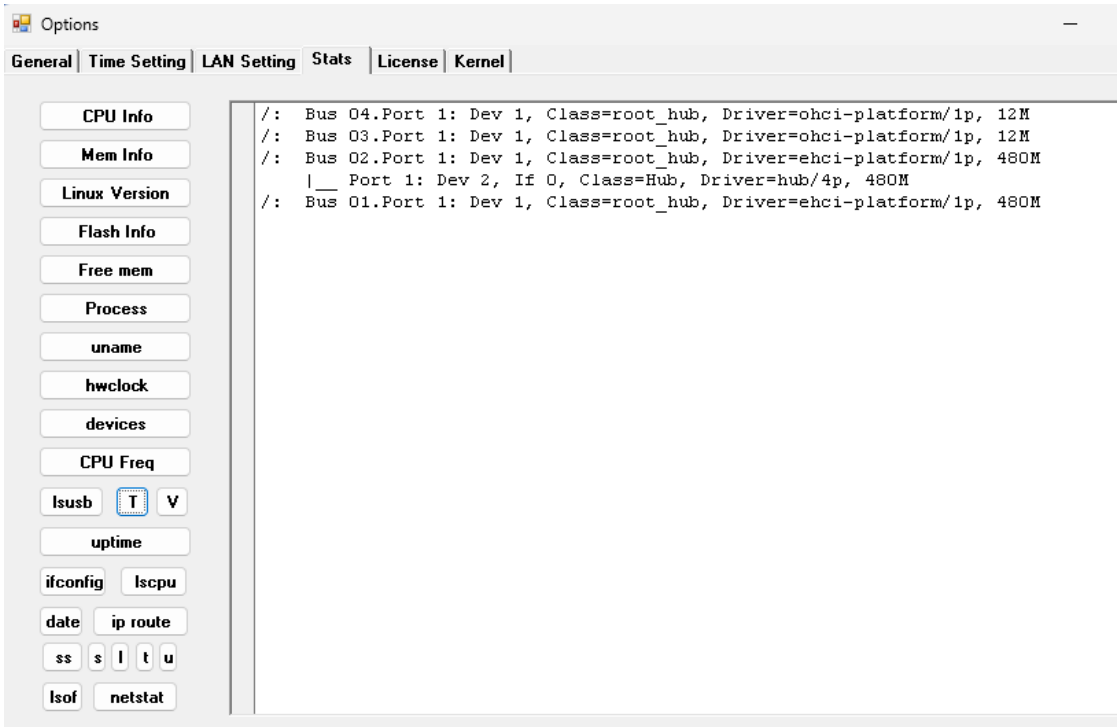
CPU Freq : Linux Command : `cpufreq-info`



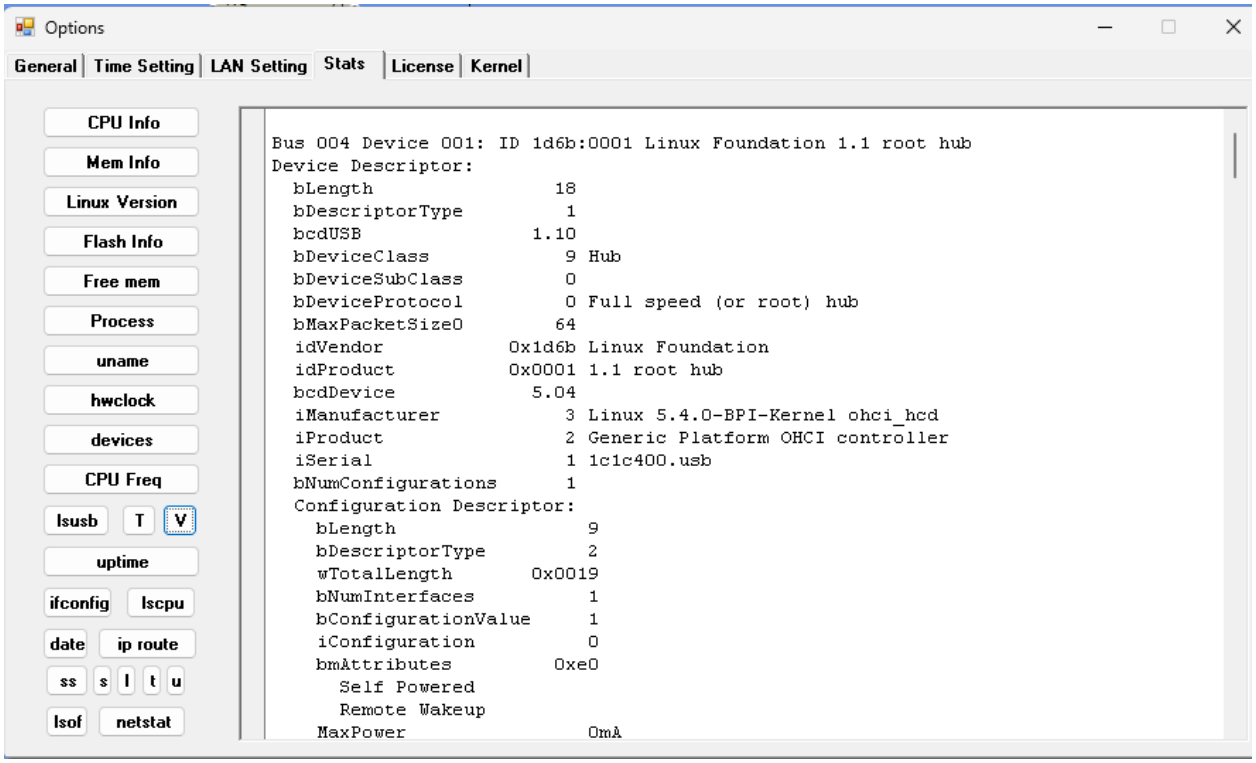
Lsusb : linux command : `lsusb`



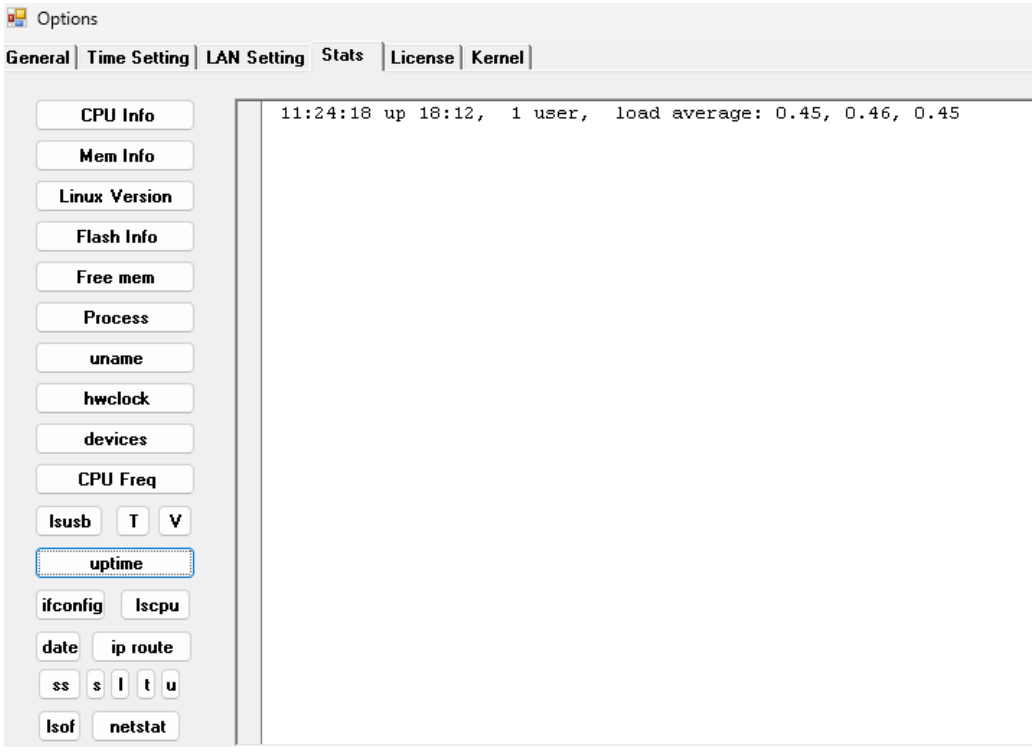
T : linux command : `lsusb -t`



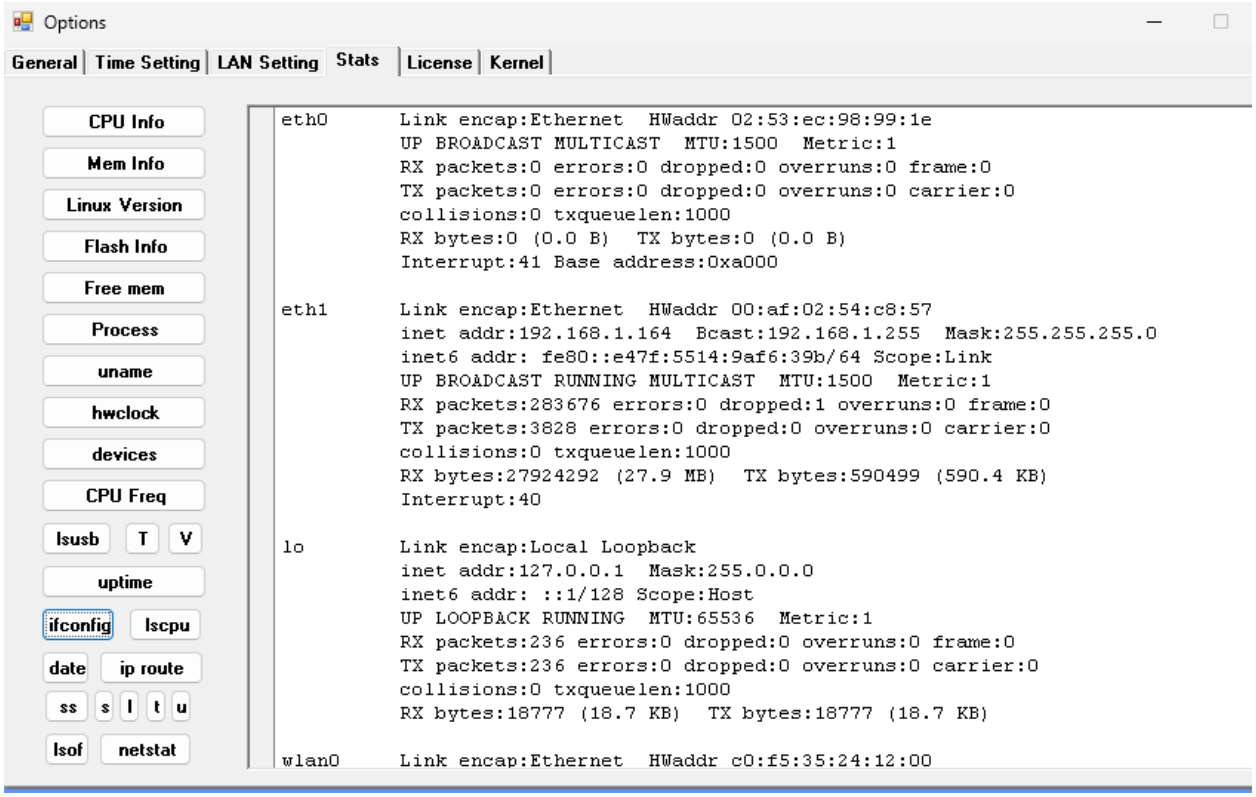
V : linux command : `lsusb -v`



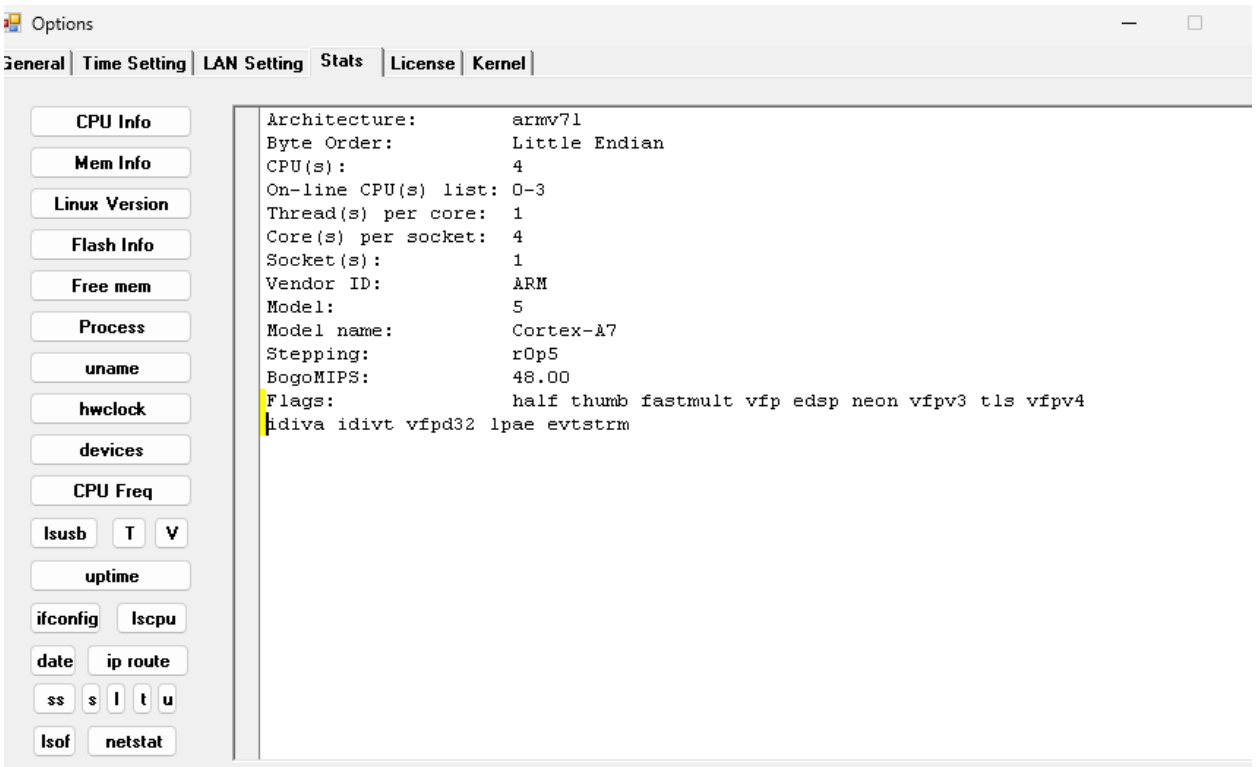
Uptime : linux Command : `uptime`



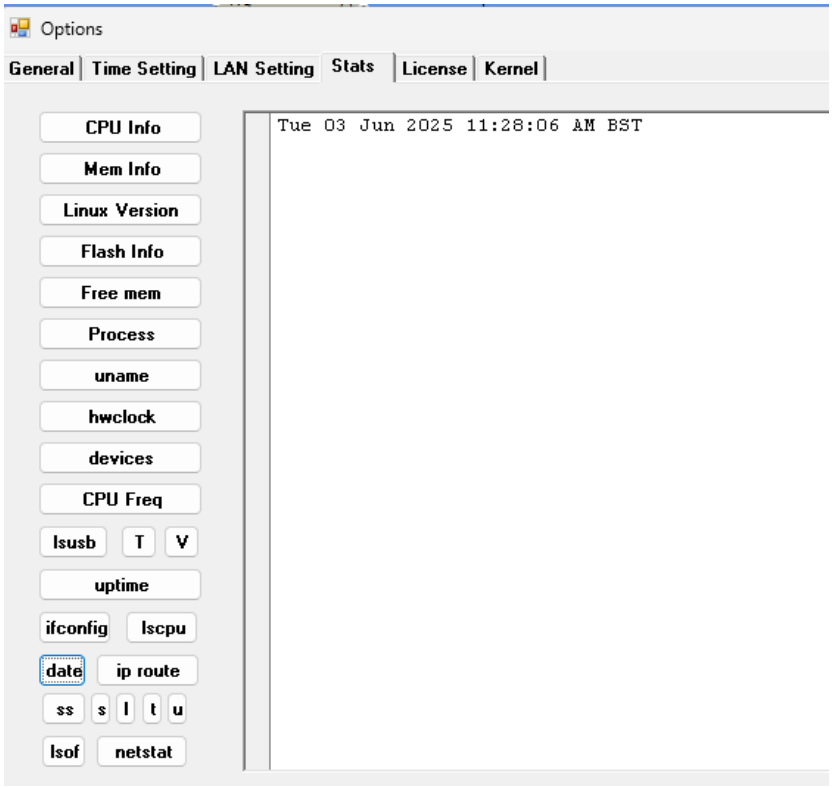
Ifconfig : linux command : `ifconfig`



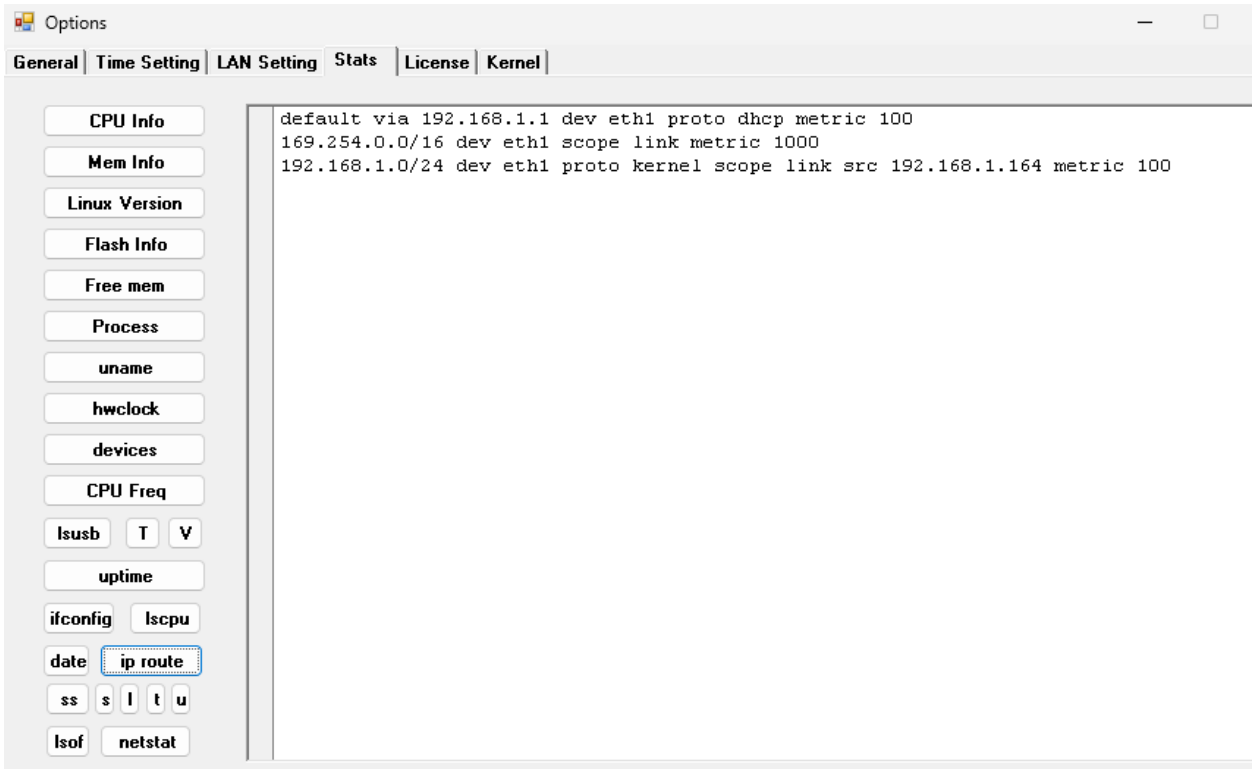
Lscpu : linux command : `lscpu`



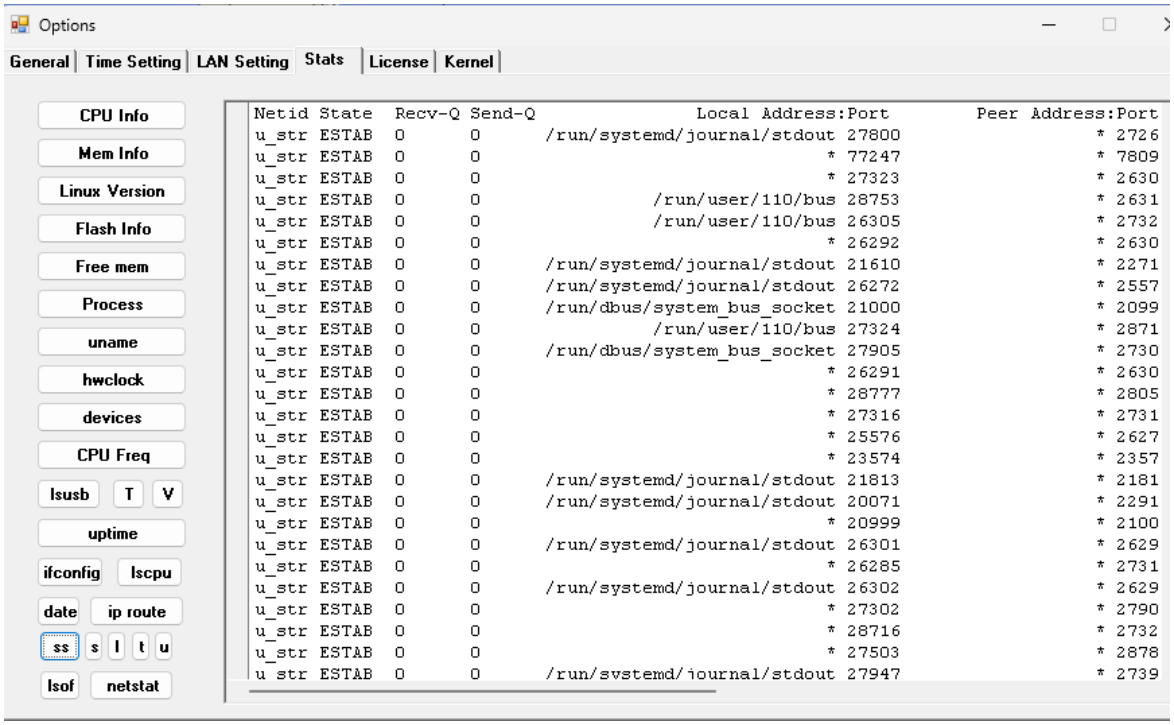
Date : linux command : `date`



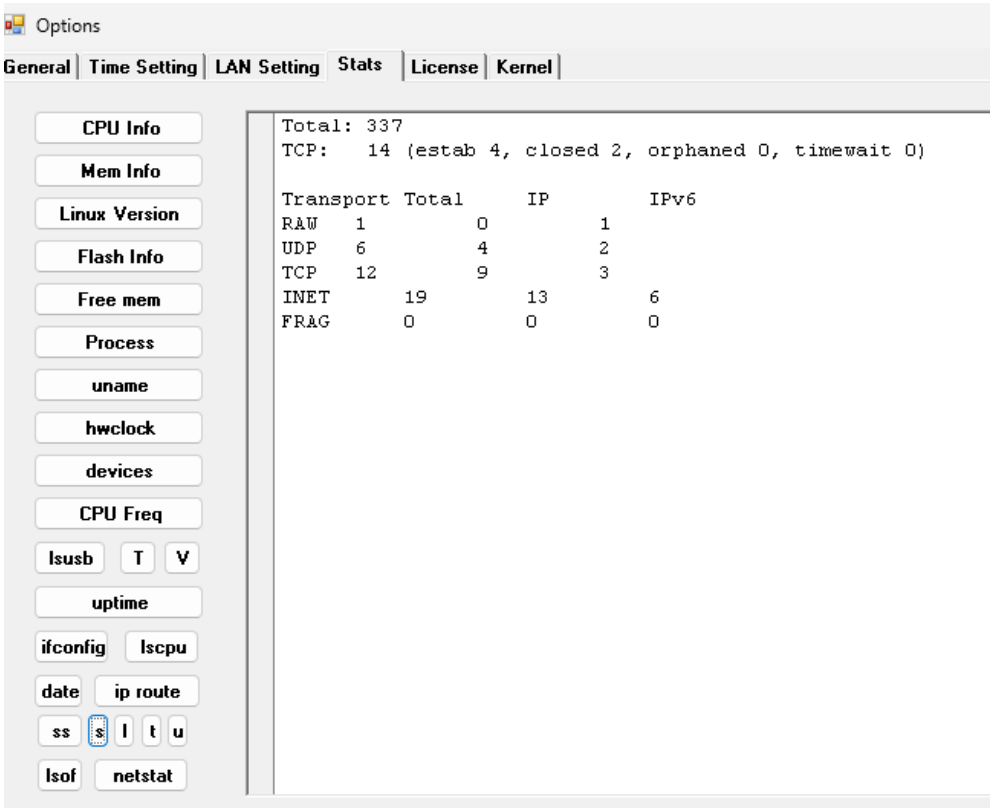
Iproute : linux command : `ip route`



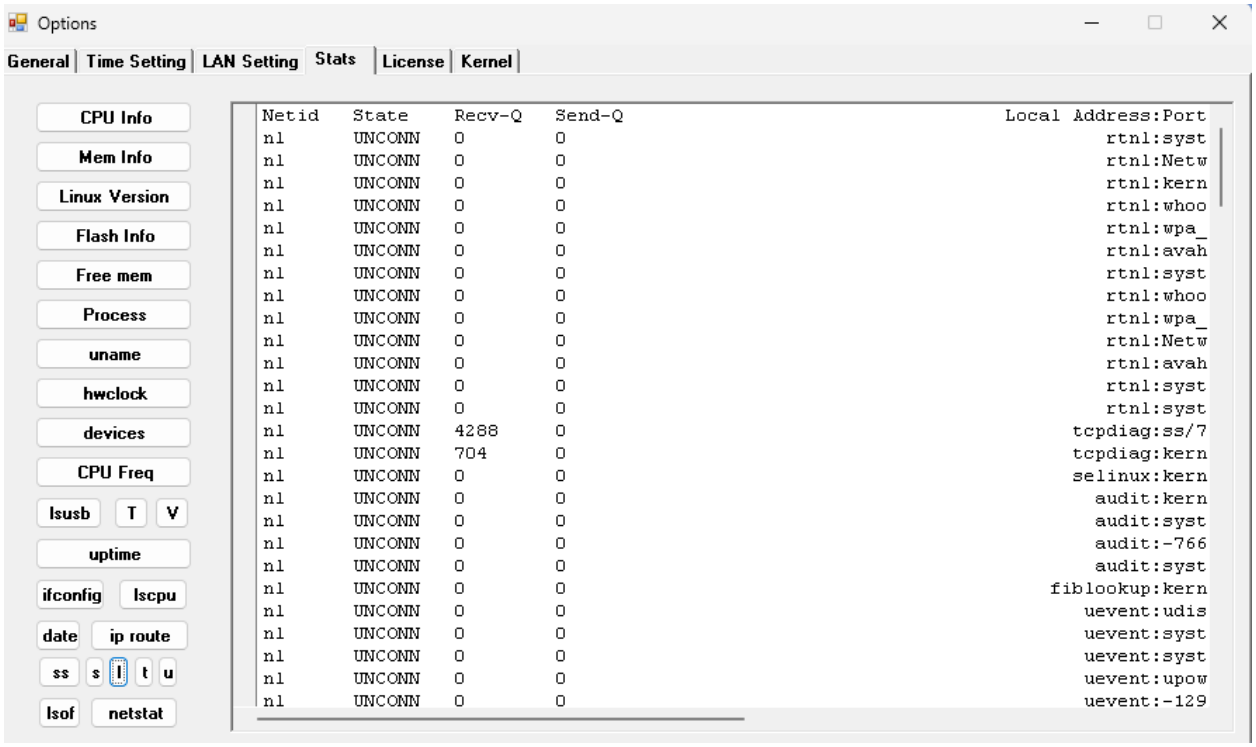
Ss : linux command : ss



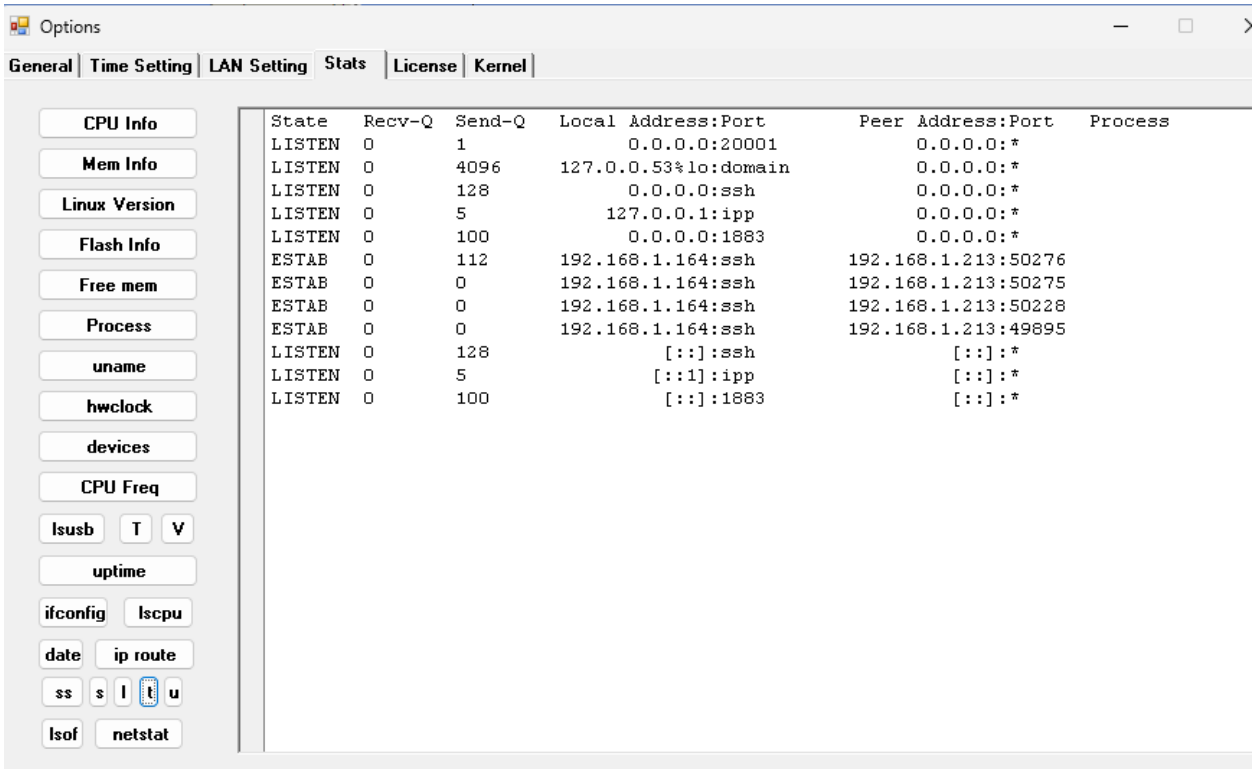
ss-s : linux command : ss -s



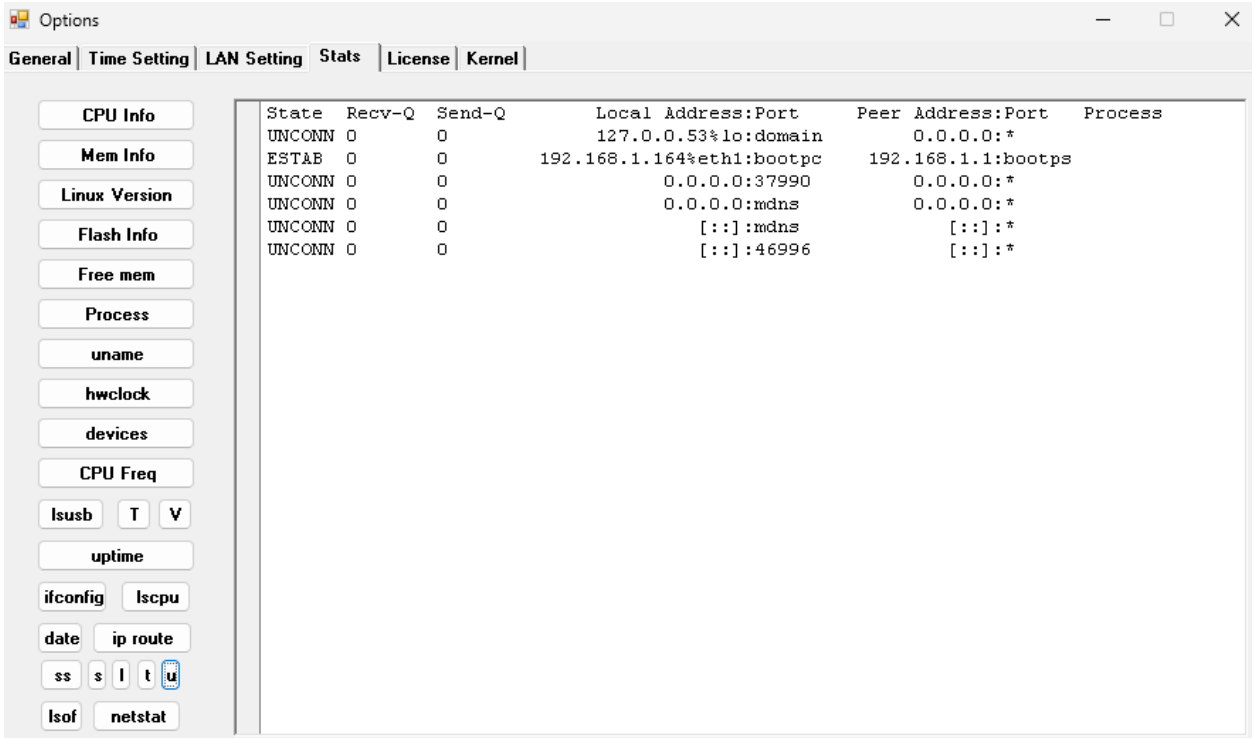
Ss – l : linux command : "ss -l



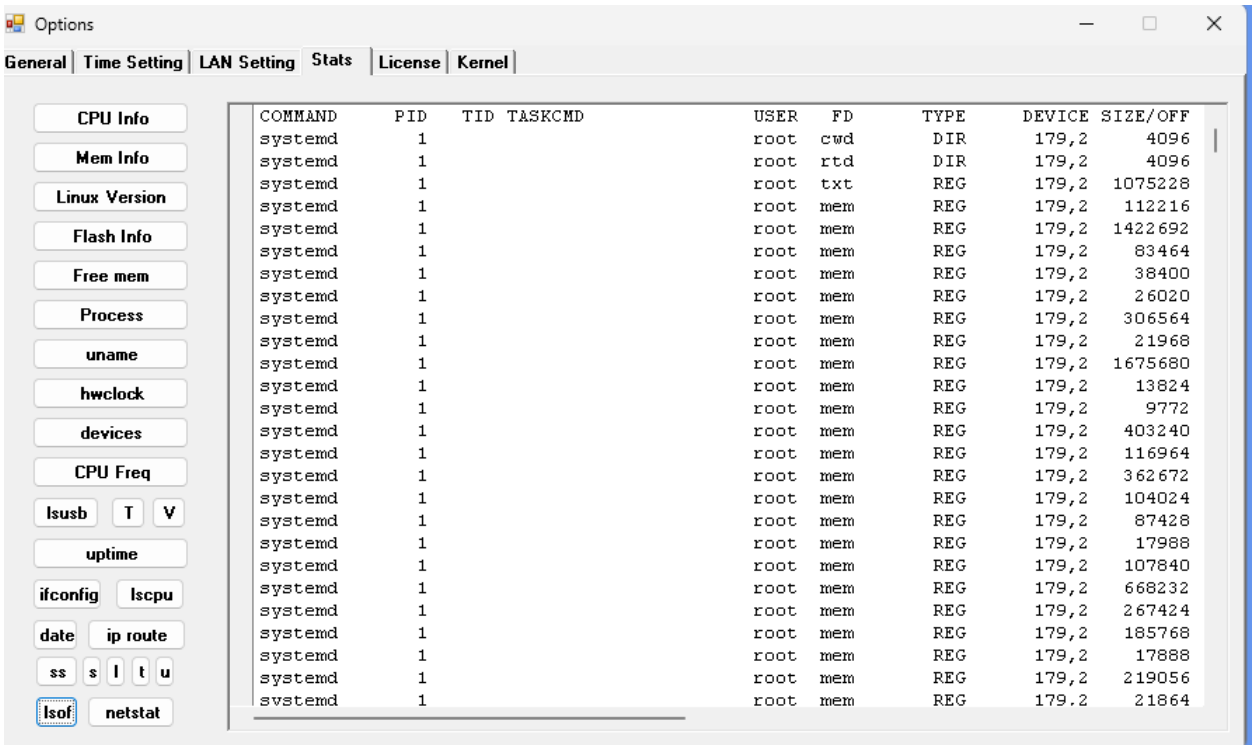
Ss –t : linux command : ss -t -a



Ss –u : linux command : `ss -u -a`



Lsof : linux command : `lsof`



Netstate: Linux command : `netstat`