TASS

Trigger and Acqusition System Simulator

An interactive graphical tool for DAQ and trigger design

Vers. 4.5

D. De Pedis - INFN Roma1 and Top1.it

Tass program is available for free on web site

www.top1.it/tass **e-mail tass@top1.it** **Oct 2023**

Elaboration of cover images by Dr Valentina Cairo, UNICAL - Italy

To my sons Gaia and Flavio
which their love highlight
my life.

TASS SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED, TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON INFRINGEMENT.

TASS

Trigger and Acquisition System Simulator

The bridge in HEP software simulation

CONTENTS

1. Overview of vers. 4.5

TASS - Trigger and Acquisition System Simulator - is a program devoted to experimental physicists developing their trigger systems and to the students learning the fundamental aspects of this job.

In the present version 4.5 several new features have been implemented in TASS program.

The most important new feature is the capability for users to develop themselves the simulation their own modules writing code in fast and easy way. For this goal TASS gives support by :

- A library of ECC (Electronic Component Control) like connectors, switches, knobs etc. This enables the user to build up the module panels in standard and fast way.
- A Device Editor allows to write most of code (~70%) in automatic way : it scans the module panel to check which ECC are present then writes directly the corresponding code routines. Only the specific behaviour of modules is responsibility of the user.
- Using the ECC library and the Device Editor the user can build himself any modules and link it to TASS program in easy and quick way (the Microsoft Visual Basic 6 package must be installed on the computer).
- The input pulses can now be produced either by the wave generator or coming from an external input file. Connection with GEANT output becomes possible.
- Input pulses can be collected via network line (Tcp/Ip) sent from the real Daq computer. This enables TASS to be used as monitor to check the hardware failure of real system.
- Break points can now be set either on input or output plug.
- Improved supports for Camac and Vme calls.
- The HBook library has been expanded with new features.
- Ecl plugs strip connectable by 3M like flat cable in fast and easy way.
- Complete rebuild of "Control room layout manager" representing the counting room.

In the downloaded package, you will find some examples showing the main features of the program and the way to build and run your own trigger system using the available modules.

For any questions, suggestions and/or bugs report please send mail to

tass@top1.it

We hope to build up an internet discussion group where people can exchange experience and suggestions and where to make available to the physicist community the homemade developed modules and ECC, widening their number and type.

Why TASS is written in Visual Basic 6 ?

Before starting to develop TASS core (end of the last century) I studied all the computer languages on the market in that time because I was well aware of the difficulties and dimensions of this job, for now TASS is about 1 million of programming lines. Of course, in the future, the implementation of new modules appearing on the market will increase that size.

In any case, especially from a graphical point of view, no competing language could do better than Visual Basic.

On the other hand, contrary to what many programmers think, Visual Basic 6 is a powerful and complete object oriented language (OOL) and, moreover, its grammar is clear, well structured and easy to use. No weird characters like (; { } [ ] etc.) typical of Java and C++, often hard to find on the keyboard, are used.

Its IDE, I think, is in absolute the best among all those on the market.

Yes, actually, Visual Basic has two problems:

- its name brings to mind the Basic toy of the first microprocessor systems like Spectrum ZX or similar. Obviously Visual Basic 6 is of a completely different power and consistency!

- Its cost, really high (about $600) considering the period in which it was placed on the market. Today the .NET version is free.

Colleagues and friends have often criticized the choice of Visual Basic and Windows, pointing out that in the field of elementary particle physics the standards currently in use are Java/C++ and Linux.

But it must be considered that TASS is a self-consistent program and the interaction with the outside world (Daq) is done through the exchange of ASCII strings over the Tcp/Ip protocol, which makes TASS itself and the Daq completely independent from each other.

In short, it is a situation equivalent to a web browser and internet pages... the two worlds are linked together through the HTML language.

On the other hand, all the sample programs in this manual can be edit by any word processor making their translation, in any computer languages, fast and simple.

In the present version of TASS program we show very few example of trigger system with their controlling Daq program written in other language than Visual Basic 6 and their skeleton is just in rough and simplest form. **We aspect help from users.**

Therefore....

...Enjoy using TASS

2. Introduction to TASS

The modern experiments in elementary particle and nuclear physics demand increasingly sophisticated trigger systems. Since the early years ’30 (Rossi’s coincidence) the trigger philosophy has been applied to extract from the background the interesting events. The trigger systems have become more and more complex, operating at higher and higher speed and their development committed to a team of expert people. So far, the design and construction of a trigger system has been based on the experience of its designers and the simulations always based on software programs.

On the other hand, the transfer of know how among the members responsible for trigger systems can become very difficult if based on pencil and paper and thousands of lines of program code. Therefore an interactive graphical tool becomes an essential part of the trigger design process.

Fig. 1 shows the main stream of an experiment. The upper part represents the hardware components, the corresponding software tools are drawn below. TASS aims to be the “***bridge*** “ joining the existing packages for physics simulation (i.e. Geant) and the analysis program (i.e. Root).

TASS is a simulation program with graphical and interactive interface, it reproduces in a realistic way the commercial Nim, Camac and Vme modules: both the front panel picture and the electrical and logical behaviours are simulated, see Fig. 2.

The simulation of a trigger system does not require to the user to write any lines of code, (see also "How TASS works" section) in fact:

- The user, to build up his simulated trigger set up, chooses from libraries the modules he needs, puts them in the crates, does the cable connections and run the system in simulated way.
- The user has access to any hardware and software characteristics as with real modules: he can move switches, push buttons, set Camac or Vme functions and so on.
- For each cable connection TASS computes the real length (translate to the real world) of the cable and takes into account its delay.
- TASS provides also many tools usually found in a counting room: digital scope, wave generator, voltmeter, Camac manual crate controller, screwdriver ...

TASS provides the capability to set hardware break points, that is, the user can pause the simulation if determined events occur. This is a powerful tool to trace the flow of signals and to evaluate the hardware dead times of system.

The capability to accept/send external signal data via Tcp/Ip network connection enables TASS to perform monitor control of a real hardware trigger set up. Indeed, by doing a cross check between the data acquired by the real trigger system and the data produced by TASS any hardware fault can be discovered in fast and easy way (see "Remote Hardware Monitor" section).

The program has been developed having in mind what the user does in the real world, e.g. it provides a sketch of control room layout where the user can place the racks and crates for all the electronics of experiment and also, for instance, the standard tools as scope, waveform generator, voltmeter and so on.

Moreover, if the user wishes to dismount a module from a crate, the program checks if any cable is still connected to it and a warning message notifies if this is not the case, and so on (see also "**How TASS works**" section).

Note:
TASS was written in Microsoft Visual Basic 6 (VB6) and therefore most of the examples shown in this manual are also written in VB6. Some examples are also presented in Java language (refer to the corresponding package in the TassSample.zip and TassRemoteDaq.zip packages).

Help from users to develop examples written in different languages will be greatly appreciated (contact tass@top1.it).

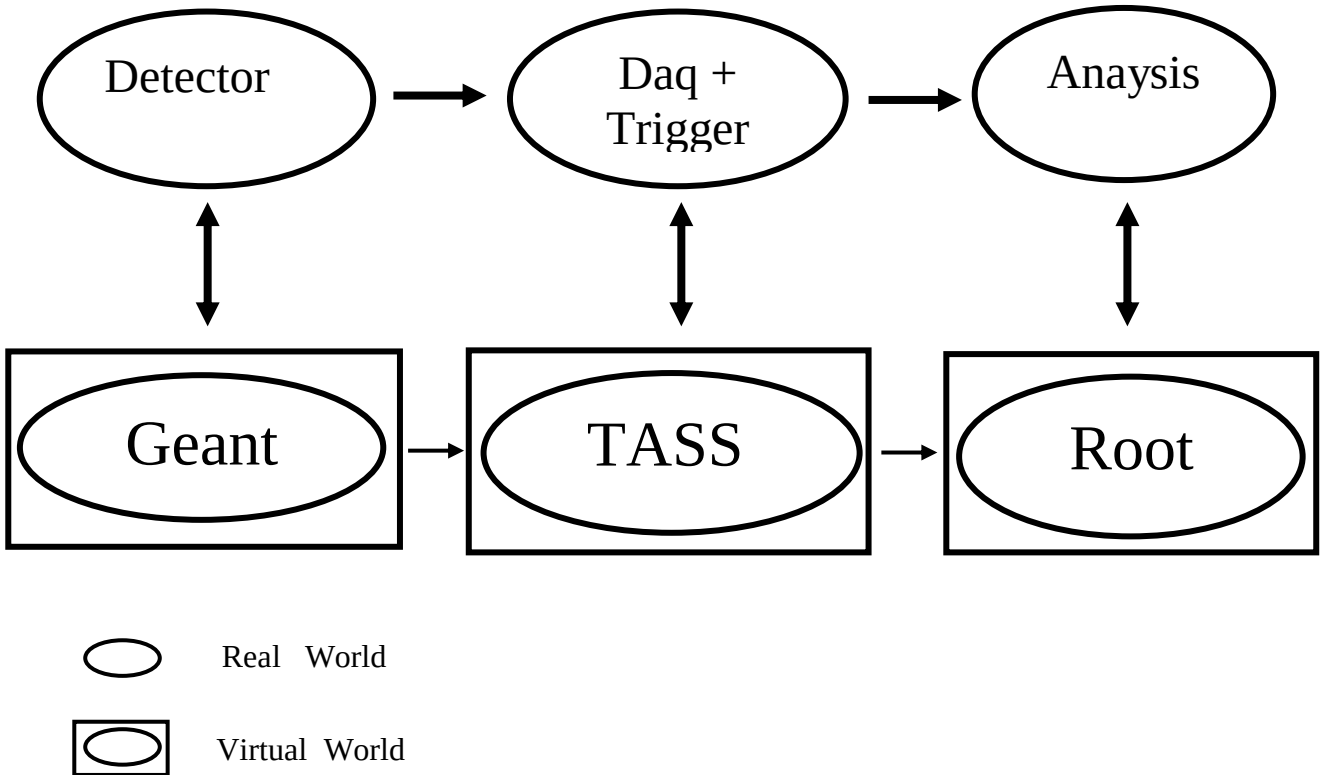


Fig. 1

TASS has been designed to be used in different environments:

TASS in stand-alone mode (no Daq)

In this way TASS is extremely useful

- **to design and to document the trigger system in physics experiments**
  This is the main TASS goal. The trigger designer can gain a lot of help building his system in virtual environment. He can choose from the library the modules he needs, put them in the crates, do the connections and run the system in simulated way.
  The designer can set and tune in an interactive way any parameter of each module. For example, he can stimulate the system sending signals by a waveform generator then he can see, in real time, the result of any change using the digital scope and measure the output voltage by a voltmeter.

- **to debug real trigger systems**
  TASS is extremely useful to identify a bad working module in an already running trigger system doing the cross check between the real set up and the simulated one.

- **to evaluate the features of new home-made modules**
  The design of a new home-made module can be simpler if the environment where it will be inserted has been simulated in advance. Good indications can arise and a better design can be done, avoiding bad surprises at installation time.

- **to evaluate the dead time and the efficiencies**
  TASS has trace of delay and duration time of signals of any module, cables connection and so on. Stimulating the input with different patterns can be easily performed an evaluation of efficiency and hardware dead time.

- **to save beam time for trigger set up and tuning**
  Most of beam time spent to tune the real system can be saved if any critical part has been simulated in advance. This can be extremely important in case of short beam allocation period.

- **to test the Data acquisition program**
  TASS provides a full set of Esone and Vme routines. The writing and debugging of Daq program can be simulated and tested gaining in efficiency and time.

- **to teach the fundaments of trigger systems**
  TASS offers to students and instructors the ability to quickly design and test a general purpose systems without any real available modules saving, therefore, a lot of time and money.

- **to understand how the modules work**
  Using mouse and keyboard the user can investigate the features of the modules without the request of expensive laboratory's tools.

- **TASS output**
  TASS supplies automatically:
  -The picture of whole layout of the trigger system (counting room)
  -The picture of any crate
  -List of used crates
  -List of used modules
  -List of used cables
  -List of connections

TASS in connection to real DAQ computer

All the cases listed above concern pure software simulations; TASS allows overcoming this limitation, in fact:

- The full integration with the real Daq program can be easily achieved by the exchange of Camac and/or Vme command via Tcp/Ip communication, that is, let TASS simulate the trigger hardware while the DAQ program runs in real world using the final computer and the preferred operating system, see "**Remote Daq**" section.

- TASS can accept external input signal data and send output data via a Tcp/Ip network connection. This new feature promotes TASS to become not only a simulator but also a very powerful monitor of hardware trigger system (see "**Remote Hardware Monitor**" section).

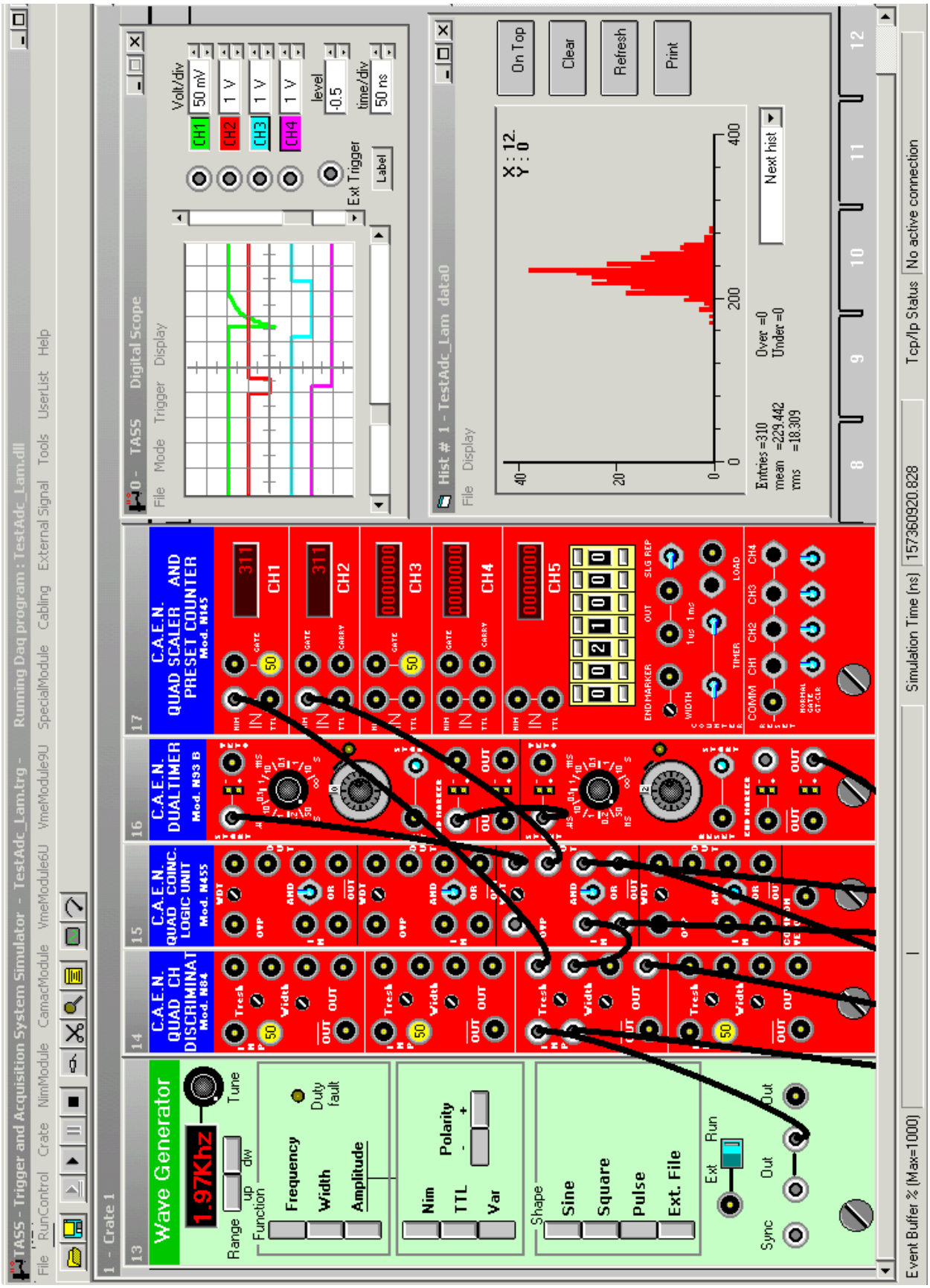


Fig. 2 partial view of small set up

3. Installing TASS

The full TASS package is free available on web sites:

http://www.top1.it/tass

TASS has been designed to run under Windows (all 32/64 bit version). Due to its characteristics, it demands a lot of computer resources, it is therefore advisable to use a suitable computer configuration.

The minimal suggested configuration is:

CPU	1 GHz or higher
Memory	8 GBytes or more
High resolution screen	1152 * 864 pixels , 17 inch (more than 24 inch preferred) (1600*1200 pixels preferred)
Hard disk	20 Gbytes free on disk
Fast network connection	

For comfortable download the TASS package has been divided into seven zipped files:

- **TassProject.zip**
- **TassDeviceEditor.zip**
- **TassModuleSource.zip**
- **TassEccSource.zip**
- **TassSample.zip**
- **TassRemoteDaq.zip**
- **TassManual.zip**

Before to download the zipped files we strongly suggest to prepare a folder called "TassUnzip" where to unzip any files concerning the Tass package.

A short description about the zipped files follows.

TassProject.zip

This is the main file containing all the program and support tools to build and run simulation trigger set up. After download, unzip the file in TassUnzip folder, then double click on Setup.exe to start the installation procedure. The following packages will be installed:

- TassProject system
- TassComponentsVers31
- HbookLib
- All the Nim, Camac and Wme modules developed so far (see the list below)

- The Tass manual
- A set of sample set up trigger systems used in this manual

TassDeviceEditor.zip

This is the file containing the program and support tools to build your own module's simulation program. You need to download and install this package only if you plan to develop code for new or not available modules (the Microsoft Visual Basic 6 package MUST be installed on the computer).

After download, unzip the file in "TassUnzip\TassDeviceEditor" folder then double click on Setup.exe to start the installation procedure. During installation process you could get some messages warning you about files already installed, answer "**Yes**" to any questions. This avoids to overwrite files installed by previous TassProject installation.

After installation is completed, check in the folder …\VB98\Template\Projects if the file "Tass Device Editor Wizard.vbz" is present, if not, make there a copy from "Program Files\Tass" folder.

If you develop any new modules and you plan to make them available to the physicist community please send e-mail to tass@top1.it for the publication on this site.

TassModuleSource.zip

This is the file containing the source program code of the Nim, Camac and Vme modules developed so far, see appendix B for a list of available modules.

You need to download this package only if you plan to develop code or you like understand how the TASS simulation of modules works.

After download, unzip the package in the folder "Program Files\Tass\ TassModuleSource". In this case you don't have to install anything, indeed the package is a collection of the source code that you can use as example and guide if you plan to develop new module.

If you develop any new modules and you plan to make them available to the physicist community please send e-mail to tass@top1.it for the publication on this site.

TassEccSource.zip

This is the file containing the source program code of the electronic components developed so far and used by TassProject and TassDeviceEditor packages.

You need to download this package only if you plan to develop code for new or not available ECC (Electronics Components Control).

After download, unzip the package in the folder "Program Files\Tass\TassEccSource". In this case you don't have to install anything, indeed the package is a collection of the source code that you can use as example and guide if you plan to develop new components.

If you develop any new ECC and you plan to make them available to the physicist community please send e-mail to tass@top1.it for the publication on this site.

TassSample.zip

This is the file containing the source program of small trigger set up, most of them will be already saved at TassProject installation time.

After download, unzip the package in the folder "Program Files\Tass". In this case you don't have to install anything, indeed the package is a collection of the Tass projects that you can use as an example and guide to build up your project.

If you intend to make any code available to the physicist community, please send e-mail to tass@top1.itfor the publication on this site.

TassRemoteDaq.zip

This is the file contain examples of class program for the C++, Java, LabView and Visual Basic languages. User can modify or use as starting point for Remote Daq class interface (see "Remote Tcp/Ip Comunication" section in the TassManual).

TassManual.zip

This is the file containing the Tass manual. The manual is already installed at TassProject installation time. This package allows you to have the Tass manual without the need to install the TassProject itself.

TASS in a click

Just after the installation you can have the first look of TASS program and you can exploit its features going in the following steps:

- Start TASS, in the main window select **File/Open** menu, then navigate on the system and select the **Sample\TcpIp_Protocol\RemoteDaq_Adc\RemoteDaq_Adc_Lam** folder.
- Double click on **RemoteDaq_TestAdc_Lam.trg** file. The trigger set up, as fully explained in "Example for Visual Basic RemoteDaq" section in this TassManual, will be loaded.
- Press **Run/Cont** button then move the switch on the "**Wave Generator**" module on "**Run**" position. The simulation starts and on Scope you can see the pulse, simulating photomultiplier signal, appears.
- Go back in the Windows Explorer and, in the folder **C:\Program Files(x86)\Tass\ Sample\TcpIp_Protocol\RemoteDaq_Adc\RemoteDaq_Adc_Lam**,
  double click on the Daq program **RemoteDaq_TestAdc_Lam.exe**. The small window "Remote Daq Test ADC" appears, into text insert field write "**LocalHost**" and press Ok button. The Daq program starts to collect data from the trigger simulation set up.
- Play with knobs, switch, buttons and watch the result of any changes on the scope and on the Daq window.
  Take in consideration that in this sample you have two crates (NIM and CAMAC). You can select them by Crate menu on the main TASS windows

Update and feedback

The update of TASS program will be posted on the web site www.top1.it/tass. These will include the new future version of TASS itself and the new modules and Electronic Components Control and any other sample set up.

It is strongly recommended that the user signs the *application form* (see **Help** menu) in order to receive all the update.

If you have question, comments, suggestions or you want to share your developed modules or your tips and tricks with physicist community please send mail to tass@top1.it.

4. Working with TASS

TASS is a self-consistent program, that is, it runs without any other program's support. The program has been developed having in mind what the user does in the real world. The main idea is based on built-in ‘‘library of modules’’, where each module reproduces in a realistic way the panel picture and the electrical and logical behaviour of the real one. A sophisticated GUI allows the user to push buttons, turn knobs, make cable connections, set Camac/Vme function and so on.

The user builds his virtual trigger/Daq system choosing from a library the modules wish he needs, places them into crates, makes the suitable cable connections and runs the simulation. Any parameter/component can be set/changed interactively. Input signals, provided by virtual waveform generators or external data files, can be used to stimulate the system and the resulting outputs can be shown on a virtual digital scope.

To achieve the above features TASS interacts with user via the following menus (refer to the corresponding sub-sections for more information).

TASS menu bar

File Menu

New (Ctrl+N)

Open a new “Untitled” TASS project and display the Layout form (see “Control Room Layout” section).

Open (Ctrl+O) 

Open an existing file containing a TASS project and its Daq program file, if any.

Save (Ctrl+S) 

Save the current TASS project on disk replacing the existing file, if any, with the same name. If the project name is “Untitled” the Save command has the same effect of Save as …

Save as …

Save the current TASS Project on disk asking the name of the file. An extension “.trg” will be automatically added. The project's window will take the new name.

Printer set up

Show the printer set up window.

Print (Ctrl+P)

Print the content of the front window on the default device defined by Windows OS.

Exit (Ctrl+E) 

Leaves TASS and return to Windows OS. If changes had been made in the current project, TASS asks if you want save it .

Run Control Menu

Run Daq Program (F8) 

Run the Daq Program.
Suitable for projects using programmable modules (Camac, Vme) in conjunction with their Daq programs.

Run/Cont (F5) 

Runs the current Project or continue if paused.
No Daq program will be taken in account. Suitable for project using non programmable modules (NIM).

Single Step (F6)

Advances the current run by a single event (see the “How TASS works” section)

Pause (F7) 

Pause the project if it is running

Reset Run

Reset the slot pointer at first event position (see the “How TASS works” section).
All used modules will be reset at idle state but no change on setting of any components will be done. The run will be paused.

Crate Menu

Show Control Room

Displays a virtual picture of racks in the counting room (see “Control Room Layout “ section).

Crate (updated by installed crates)

Displays number and name of used crates. Selecting a crate in the list, it will be displayed on the main TASS window.

Nim Module menu

This Menu is enabled only if TASS shows in the main window a NIM crate.
Selecting this command the user can get one of the NIM modules shown in the menu list and can put it in the crate shown in the main window. Refer to “Using Nim, Camac and Vme modules” section for deeper discussion.

Camac Module menu

This Menu is enabled only if TASS shows in the main window a Camac crate.
Selecting this command the user can get one of the Camac modules shown in the menu list and can put it in the crate shown in the main window. Refer to “Using Nim, Camac and Vme modules” section for deeper discussion.

Vme Module Menu

This Menu is enabled only if TASS shows in the main window a Vme crate.
Selecting this command the user can get one of the Vme modules shown in the menu list and can put it in the crate shown in the main window. Refer to “Using Nim, Camac and Vme modules” section for deeper discussion.

List Vme address space

lists on logger window the address space reserved by the modules installed on the current Vme crate

Cabling Menu

The items in this menu allows to make, delete, show and hide the cabling connections in the Project (see the “Cabling “ section for more details)

External Signal

Set Break Point (Ctrl+B), Remove Break Point, Remove All Break Point

Set or remove the break points.
TASS provides the capability to set hardware break points, that is, when a wished event occurs on a specific plug the program pause the execution. See “Break Point “ section for more details.

To set (remove) a break point select Set Break Point (Remove Break Point) and Click on the plug where you want set (remove) the break.

Connect to input file , Remove from input file, Remove all from input file
Set or remove signal issued by input file (.sif) connected to an input plug

Connect to output file, Remove from output file, Remove all from output file
Set or remove signal delivered to output file (.sof) connected to an output plug

Show message on Daq socket
Shows the messages exchanged via the TCP/IP Daq socket

Tools Menu

Digital Scope 
General purpose digital scope (see the “Module description” section)

Wave Generator
General purpose wave and function generator (see the “Module description” section)

Show Meter 
Toggle command showing the voltage from any module output plug.
Click the command then points out and hold down left mouse button on the output plug, a small light blue tip appears on the menu bar showing the measured voltage.
The same command is used to show any other variables (width, threshold etc.) see “General Consideration “ in the “Module description” section. In conjunction with the Ctrl button this avoids changing the value of the parameters associated with the knobs or screw (turn not allowed).

Refresh crate (Ctrl+R)
Force the redrawing of presente crate

Options …
Open the general options dialog window

UserList Menu

Show Logger, Clear Logger

Show or clear the "TASS Logger" window

Crate List, Module List, Cable List, Signal Connections List

Show on immediate “TASS Logger” window the corresponding list

List all

Show on immediate “TASS Logger” all previous items list.

The "TASS Logger" content can be either saved on file(the .log extension will be add automatically) or printed on printer.

Help

About TASS …

Open About TASS window where you can find the Server Local Ip number, very useful for Remote Daq connection (see **Remote Daq** section).

TASS and Device Editor Manual

Here you find this manual and the Device Editor Manual on line version

Donate

Here you find an exhortation to help the TASS developer team with a small gift.
You can imagine that the development and maintenance of TASS involves a work of thousands of hour per year. We are constantly involved in the development and coding of new modules entering in the market.
Your little financial support allows us to work harder and with more time to devote to TASS.
Moreover remember that, on request, it is also possible to develop TASS simulation for your own homemade modules, contact us at tass@top1.it

How TASS works

TASS performs a discrete simulation of the signals in a logic circuit, the program doesn't attempt to analyse the internal circuit of the modules but simply takes in account their transfer functions. For each module, the program assumes the mean values of parameters (gain, delay, width etc.) as reported in data sheets of the constructors. The delay of cables are taken in account, too. Nevertheless, no raising and falling time are taken into account for digital signals.

The analogue signals are treated in a discrete approximation, meaning that signals level and time change only in step, rather than continuously (typically the sampling rate is 2ns/step).

The simulation is **"event driven",** ***an event being a change in the amplitude level of a signal***. Beyond the standard mouse and keyboard events, TASS defines the special event "**Pulse**" usually associate to the Nim and Ecl signal plugs.

Each time an Pulse event occurs, the program computes the transfer function of all modules whose inputs are affected by that event and reflects the changes on their outputs. No computation is performed when no event occurs, so delay/width of devices setting or clock values do not affect the simulation speed. Of course all the events are handled by TASS and are completely hidden to the user.

Time values are internally stored in 64 bit using Currency format: the values can range from 0 to 922.337.203.685.477,5807 ns allowing a total simulation time more than ten days with resolution better than picoseconds !

TASS can run trigger simulation either without any external Daq program (e.g. if only non programmable modules are involved) or with a companion external Daq program that can be resident on the same computer or operate via network connection.

In both cases, the **Single Step** option is provided, that is, the program analyses a single event then the execution pauses(*).

Using Camac and/or Vme programmable modules the user has to provide an suitable external program to:

- initialize any modules
- read and/or write internal register
- check the availability of data
- write on disk files any results

Usually all these tasks are handled by, so called, external Data Acquisition (Daq) program.

TASS provides the Esone library routines to handle the Camac and Vme functions and many other software tools to enable the user to write and to link a Daq program in fast and easy way.

Usually the way in which the Daq program interacts with TASS simulator is via the Tcp/Ip internet connection, that is, the user can use his/her standard real Daq developed in the proper language and operating system, only the calls to handle the Camac and Vme functions have to be replaced with the procedure issued in the Tass package (see **Remote Daq** section).

(*)May happen, pushing the **RunControl/SingleStep (F6)** menu, apparently nothing changes but if you watch on the **Simulation Time** label on the task bar in the bottom part of main Tass window, you will found the simulation time increasing.

Control Room Layout

When user starts to design a new project simulating a trigger system, first of all, he has to inform TASS about the physical disposition of the real trigger set up. For this purpose TASS provides a virtual layout of the counting room.

User must choose a racks/crates configuration as much as possible similar to the real set up because the relative position of the racks/crates will be scaled to real world and used to compute the real length of cable connections and their delay.

To build up a new project select in TASS main window the **File/New** menu, the "Control room layout manager" window appears. There, select **Project/New Project** menu and choose the suitable racks configuration, TASS allocates the racks in a predefined position. Any time the user can add more racks by **Racks/Add** menu. To remove a rack select **Racks/Delete** menu then click on the rack you want remove, only empty racks (see later) can be removed(*).

The location of each rack can be rearranged dragging it to the whished position, see fig.3. An automatic realignment can be done by the **Rack/Align Racks** menu.

In order to arrange the rack configuration as much as possible similar to the real set up the relative position of any rack (in cm), respect to the "Start of floor baseline", is shown in top.

If more room to allocate the racks is needed, you can resize the "Control room layout manager" window dragging its right edge.

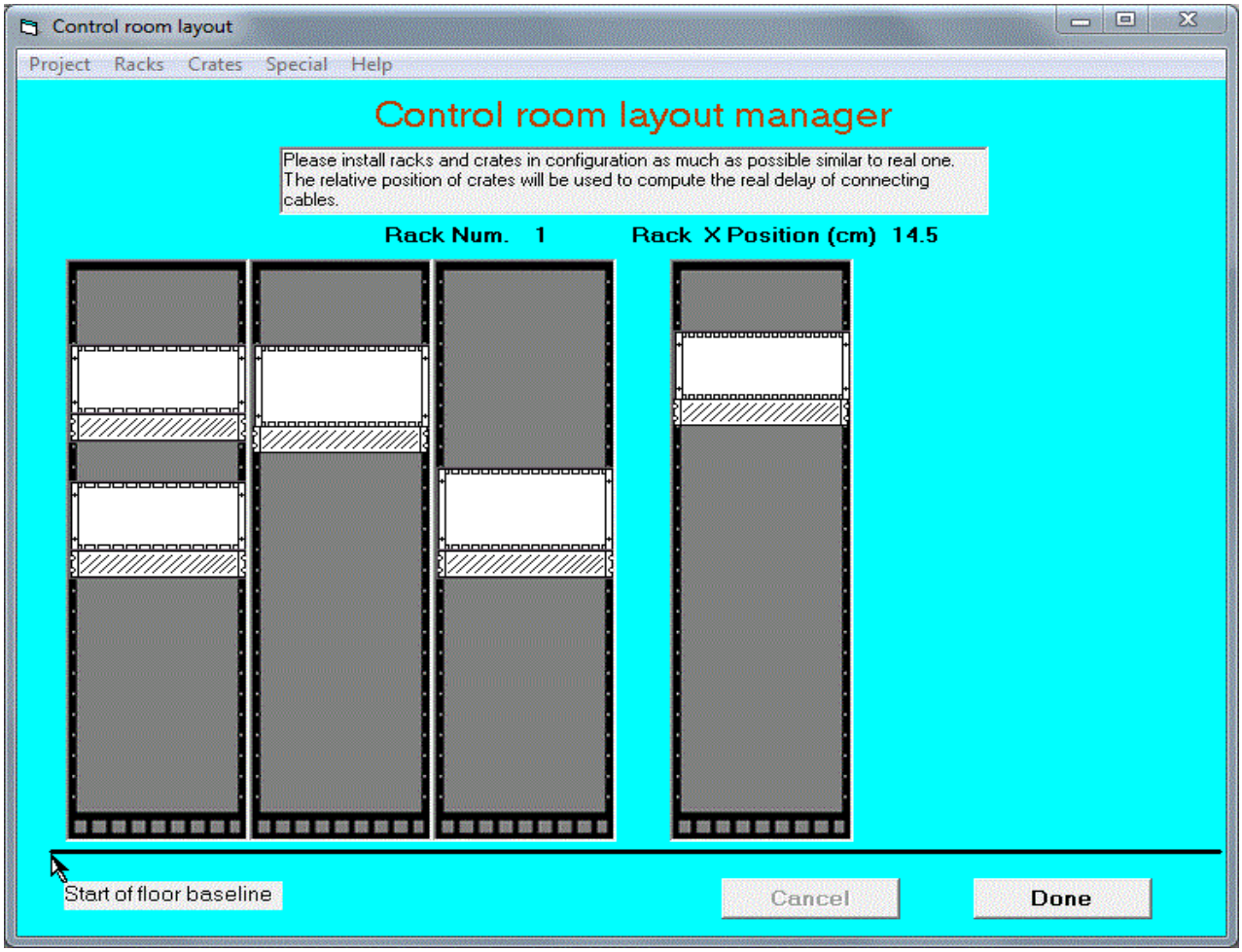


Fig.3

(*) For now, only racks with consecutive numbers are allowed

When racks are in the right place, you can fill them with the crates. You choose from **Crate** menu the Nim, Camac, Vme6U crates you want install. Drag and drop the selected crate everywhere inside the rack, a small picture, representing the chosen crate, will be allocated perfectly aligned inside the rack. You can move the crate along the vertical direction either by dragging it or by right click inside the crate itself and then **Move Up/Move Down** in the popup menu, (fig. 4).

Pay attention, the crates can be allocate only at predefined positions corresponding to the screw holes in the real racks, this feature preserve you to choose a not realistic configuration.

To remove an installed crate use either **Crates/Remove** menu and then left click on the crate you want remove or the corresponding item on the popup menu (fig. 4).

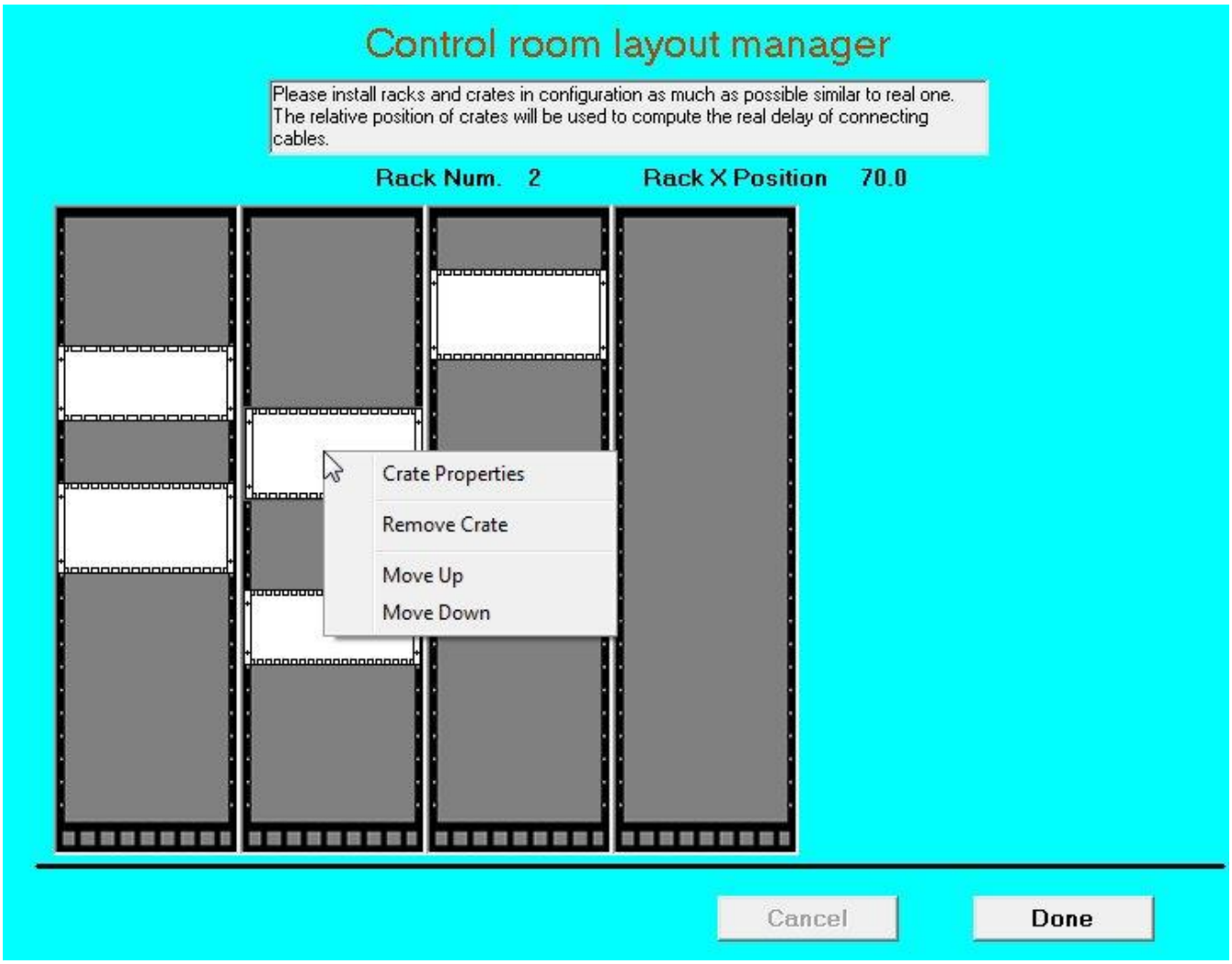


Fig. 4

User can install the fan units and patch panels, too. Select them from **Special** and **Special/Patch Panel** menu respectively and refer to the above description concerning the crates for corresponding features.

To display the main characteristics of each installed crate select **Crates/Show Crate Properties**, the following window is displayed (see Fig. 5). In the field Name the user can insert any name to identify the crate, that name will appear in the Crate menu of main TASS window.

Note : <u>the numeric digit appearing on field Name is not editable and is under TASS control.</u>

The X Position represents the distance respect to the left edge of floor baseline in the "Control Room Layout Manager" (Fig. 4) and the Y Position represent the height respect to the top of parent rack.

Installation of Camac crate include automatically the installation of its crate controller in station 24-25 (the standard "TypeA" used). The Branch and Crate number must be set (Fig. 5) and their combination must be unique in whole the system, in case of address conflict a warning message is issued.

Installation of Vme6U crate include automatically the installation of its crate controller in station 1 (the standard "CAEN_V1718LC" used).

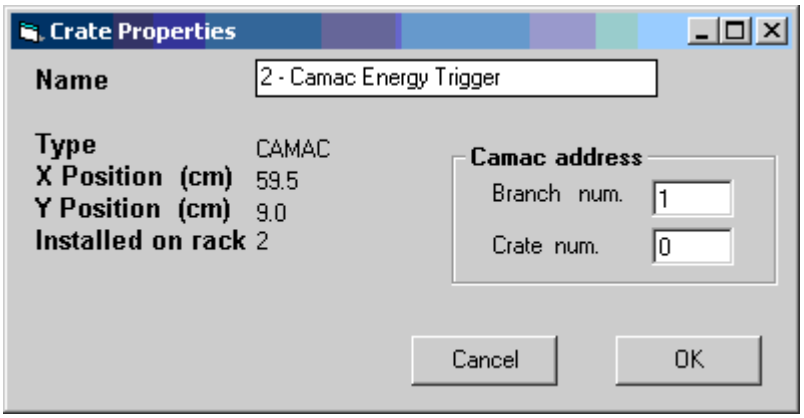


Fig. 5

When any suitable crates are installed, click on the **Done** button of Fig. 4, Tass will go back to the main windows showing the first crate. In the main window, user can navigate over the whole set of installed crates selecting the **Crate** menu.

Using Nim, Camac and Vme modules

The registered modules are collected in three different menus depending on their type (NimModule, CamacModule, Vme6UModule)(*). Only the menu corresponding to the crate currently shown on the main window is enabled, this feature prevents the possibility to install a module in an inappropriate crate.

At start time, TASS fills each menu reading the module list from TASS.ini file located in the {WinSysPath}/TASS(**) folder, this file can be edited by any text editor. The Tass.ini file will be update when you install new module code or when you develop your own module by the Tass Device Editor package.

To use a module in a simulation the user selects it from its menu, the cursor changes to four-arrows shape, then click anywhere in the wished station inside the crate, TASS takes cure to allocate it correctly aligned. TASS also checks if there is enough room in the corresponding crate to allocate the module and, if not the case, gives a warning message.

The last selected module is directly available holding down the Cntr key, this feature allows a fast filling in case you need several equal modules installed in the same crate.

To remove an installed module click on the screw (on the extractor handler in Vme module) in below part of panel. Only modules without any cable connection can be removed.

CAMAC vs Vme standard (see also RemoteDaq Programming Model section)

CAMAC (Computer Automated Measurement and Control)

is a standardization system for control electronics and data acquisition. It was developed in the 1970s as a common platform for building electronic equipment for scientific research, particularly in particle physics.

Although CAMAC is currently a disused system and by now no manufacturer would have an interest in placing new modules for this standard on the market, however it should be kept in mind that in many laboratories around the world modules of this system are still in use, and this is reasons TASS supports such a standard.

A CAMAC system consists of a controller (usually a computer) that controls and communicates with the CAMAC modules all housed in a container called "Crate". CAMAC uses a parallel bus for communication between modules and offers a number of mechanical and electrical standards for interface modules.

In the CAMAC system, the internal registers of the modules are addressed using a hierarchical system of addresses and sub-addresses. Each module is installed in a specific slot of the crate and can have one or more sub-addresses, which represent the different registers or channels within the module itself. To address an internal register of a CAMAC module, it is therefore necessary to specify the crate number (C), the module slot number (N), the sub-addresses (A) of the desired.

(*) The old Vme9U standard is no anymore supported

(**) The {WinSysPath} is the path of the operating system folder, usually is the “C:\Windows\SysWOW64\” folder

register and the function (F) that you want to operate. The entire command is known by the acronym CNAF

In the 1970s, the ESONE Commission (European Standards on Nuclear Electronics) issued the EUR4100 and EUR4600 specifications relating to the CAMAC system(*)

Vme (Versa Module Eurocard)

In the Vme system, the internal registers of the modules are addressed using a system based on the base address (unique for each module) and the offset which specifies the register within the module itself. Each Vme register therefore has an address formed by (base + offset) which uniquely identifies it within the Vme system. In this way, the Vme system allows you to flexibly access the internal registers of the modules.

In TASS, for Vme multicrates systems, all the addressing space is unique, that is, no overlap of base addresses for all modules is allowed. List of base addresses of all installed modules can be obtained by **VmeModule6U/List Vme address space** menu.

To set or modify the Vme base address register see below section.

General considerations

Running TASS, the user has to take in account some general considerations:

- Usually the modules have internal components (jumper, trimmer and so on) to set or select different behaviour and/or to set/modify the register address of Vme modules. To access to internal components double click on the name of module in the top edge, a lateral panel will open showing the internal part (see below). A Close button provides to close the lateral panel. The back panel components are accessed in the same way.
- Due to digital behaviour of the scope, after any change on the project set up, the user has to wait at least one full trigger cycle to have the updated display. The label named " trg'd " on menu bar of scope is highlighted when trigger starts and hide when display is updated.
- The scope and the Cable Info window stay always "on top" (that is, they cover any other behind pictures). If they loose this property, click the corresponding button on the menu bar.
- TASS never shows the cables connecting the scope's plugs. That cables are always 8 ns long. Use the Cable Info tool, instead of (see next section).
- The flashing light (led or lamp) on the modules doesn't reproduce the real on/off duty cycle.
- The by-passed coupled inputs (that is, inputs having high impedance and internal short cut) are treated as one input and one output plug. At installation time, TASS always shows this output plug with a plugged 50 ohm terminator.
- The settable or movable components (button, switch, knob and so on) show a Hand cursor, when the cursor is over the component itself, see the Cursor summary in section 4.6,
- The screws to set threshold and width show a Screwdriver cursor as well.
- The cursor doesn't change its shape when it pass over the input and output plugs but it changes its label in agreement to the selected cabling procedure (see Cabling section).

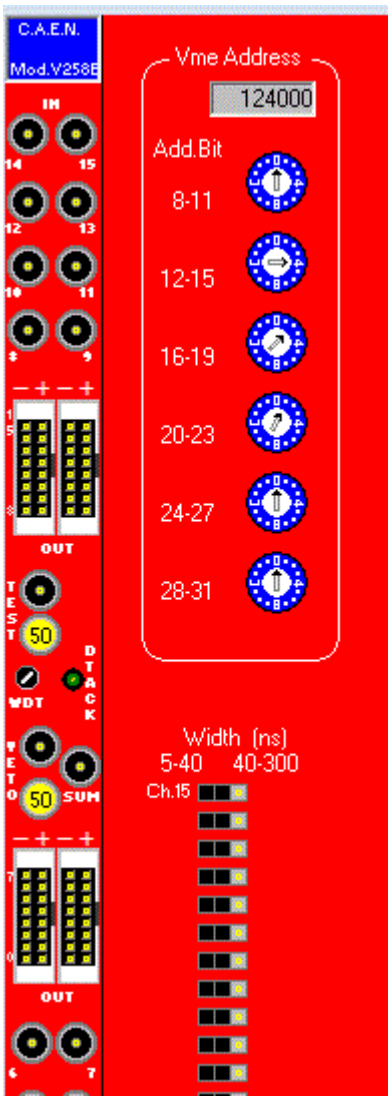
(*) see https://core.ac.uk /download/pdf/132578818.pdf and also William R. Leo Techniques for Nuclear and Particle Physics Experiments, ISBN 0-387-17386-2 Springer-Verlag.

- The knobs and screws can be turned clockwise by the mouse left button and counter clockwise by the right button.
- The three position switch moves in round fashion (i.e. up, centre, down, up….) when clicked.
- The monostable push buttons are pressed when the mouse left button is down and released when it is up. Usually the corresponding action is performed when the button is released.
- The value for threshold, offset, width ecc. are shown by small tip on the menu bar. To show the tip select **Show Meter** from **Tools** menu or press the corresponding button.
- The tip's color corresponds to different types of measurements in agreement to following table:

Color	*measurement*	*unit*
Yellow	width of pulse	ns
Green	threshold	V
Light blue	output voltage	V
Gray	offset	V
Violet	special	spec

- Use Option … menu to set different behaviour than the default one.

Double click to open the internal components panel (Vme partial view)



Cabling

TASS provides a full set of functions to do the cabling procedure in a fast and easy way. The user can make, show and delete the cables in single, multiple and all system connections basis.

By default TASS doesn't show the cables of the connections to avoid to hide the module panels in background (in real world you can move the cables to have access to module's components, but you can't on a picture screen). Of course, the user can modify the default setting choosing different option (see Option section).

Note: Cables connecting the scope never are shown and always they are 8 ns long.

Make connections (Ctrl+M) 

To make the connections between plugs you have to select Make Single or Make Any from Cabling menu. The cursor changes to *Make cursor* (see later 4.6 *Cursor summary* section) then you click on the first plug you want to connect. The plug itself changes style to notify *busy plug* and cursor changes to *Make+ cursor* meaning pending connection in progress. Point the second plug and click again the mouse, the connection is now established, the length and delay of cable is automatically computed (see later).

If you have chosen Make Any the cursor returns to *Make cursor*, and you can make a new connection, otherwise it changes to standard cursor.

If the plugs to be connected are on different crates you have to navigate through them selecting the proper crate in Crate menu.

To erase and/or show a pending connection, click anywhere on the menu bar and answer to the questions.

In any case the program checks the consistency of type of plugs (Nim/Ecl) and their functionality (input or output) and notifies by a message an incompatible connection.

Nim (Lemo) connections

TASS computes the minimum cable length, scaled to real world, needed to do the connection. The assigned cable length is the longer nearest value commercially available (0.5, 1, 2, 3, 4, 5, 6, 8, 10, 12, 16 ns), for length bigger than 16 ns the computed value is assigned.

By default the digital scope, being a movable instrument, has inputs always connected by cables of 8 ns length and they are never shown.

.

Ecl connections (single line)

For single Ecl the connection procedure is similar to the above Nim connection but, in this case, TASS doesn't check for commercial length and takes in account the computed delay.

Moreover the Ecl standard allows to connect on the same line more input plugs in daisy chain and any plug can have reversed polarity.

Tass implements this features in the following way:

- The first plug of an Ecl connection chain MUST be always the output plug then, one after another, any other input plugs follow. A small red dot marks the positive pin of the plug (see *Cursor summary*).

- To reverse the polarity double click on the plug itself, the red dot moves to the other pin and the reverse polarity of signal is provided.

Delete connections (Ctrl+D) 

There are four possible options to delete existing connections: **Delete Single, Delete Any, Delete in Module, Delete All.**

- **Delete Single** or **Delete Any** changes the cursor to *Del cursor* (see below) and click on one of the plugs holding the connection you want drop. If Delete Any has been selected then you can continue to drop other connections, otherwise the cursor returns to *standard* cursor.
- **Delete in Module** is mainly used when you want to dismount a module from the crate (you can not remove a module if any plugs is still connected). Click on any plug belonging to the module then all the connections on that module will be dropped.
- **Delete All** is useful if you want start from scratch the cabling job, a warning message will ask you to confirm this choice.

Nim (Lemo) connections

The user points one of two plugs of connection he want delete and clicks the mouse. The logical connection is dropped and the plugs return to *free plug* style.

Ecl connections (single line)

If the user clicks on the output plug then all the Ecl connection chain will be dropped.

If the user clicks on an input plug then the connection between the selected plug and the previous one plus any connections between the selected plug and the last plug in the chain will be dropped.

Show connections (Ctrl+V) 

The user can ask TASS to selectively show the cables by Show Single, Show Any, Show All options. In this case the cursor changes to *Show cursor* (see below) and cables will be drawn by black line for Nim connections and grey line for Ecl connections.

The user can choose to show connections by straight lines or by more realistic flex lines. In both cases the delay length of cables itself will be the computed one (see later section).

TASS never shows cables connecting scope plugs.

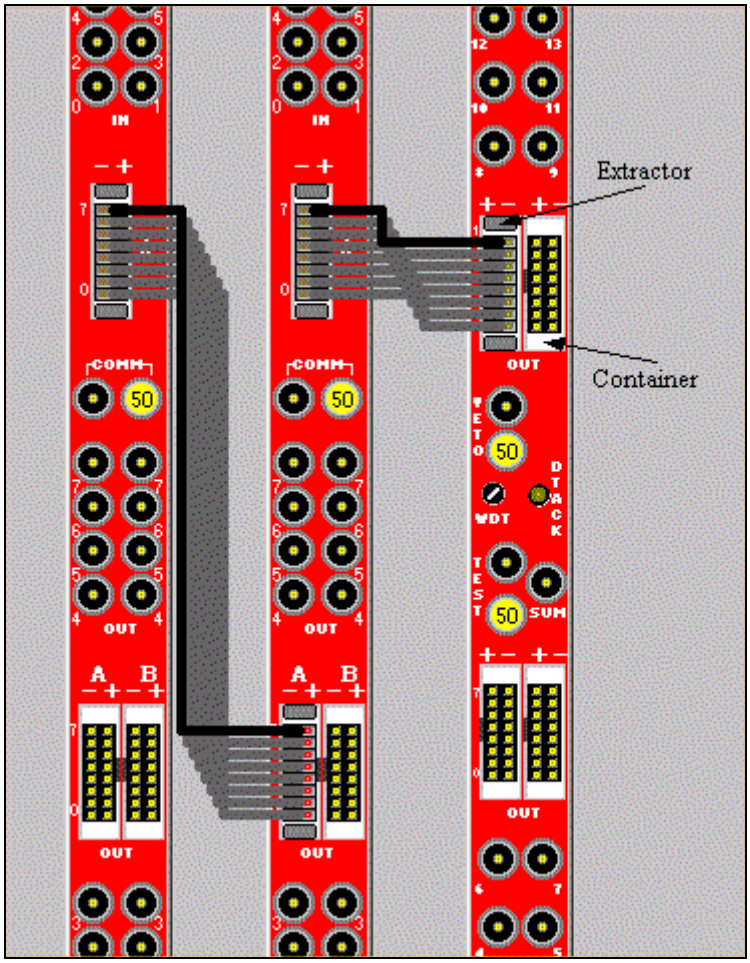
Hide connections (Ctrl+H)

Selecting this item TASS hides the cables of whole system but any logical connections and their attributes still work.

Hiding cables makes faster the panel redrawing process. Use this option when you are navigating across the crate.

Ecl flat cable connection

Usually the Ecl connections on the multipins plug are done using flat cable bundle (3M type). TASS gives support to do this job in easy and fast way (see figure).



To do a bundle connection click on the container of the output plug then on the container of input plug, TASS takes cure to realize the connections for all pins. Repeat this procedure to do a daisy chain connection. A container when clicked show the closed extractor (black part).

To remove an existing connection click on the extractor of either out or input plug.

Note: It is possible connect only container having the same number of Ecl pin pairs.

Connection Info (Ctrl+I) 

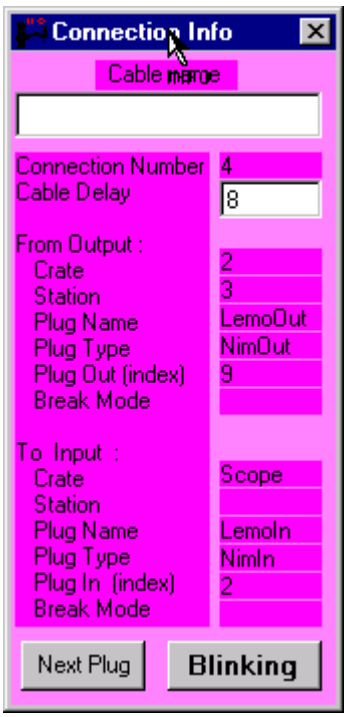
The best way to trace the cabling of whole system is using the **Connection Info** option. Selecting this item the cursor changes to *Info cursor* (see below) and a small **Connection Info** window appears over any other draw (see figure).

Clicking on a plug of the connection you are interested in, the main characteristics of connection will be reported.

The user can assign a label to the selected connection writing its name in Cable name field.

The Cable Delay field shows the default computed length (in ns) [*] as the minimal commercial length allowing the connection.

The default length can be modified overwriting the new value on Cable Delay field, if the new value is smaller than the previous one, a warning message will ask to confirm this choice (this is useful when you foresee to use a fast cable for the connection).



The **Blinking** button causes the plugs corresponding to selected connection to flash allowing a fast and easy tracking.

The **Next Plug** button is enabled only if the plugs of the selected connection are on different crates. Clicking on this button TASS shows you directly the crate where the other plug is sit.

In **Tools\Options\Cabling** tab the following options are available:

- "Trace cable when flashing"
- "Flash whole Ecl chain"
- "Blink rate"
- "Blink number"

[*] The speed of signal in cable is computed equal to 5 ns/m.

Break Point

TASS provides the capability to break the execution of the running Project when a wished event occurs[1]. This feature allows the user to trace and to analyse step by step the signals flow.

To set (remove) a break point select **Set Break Point** (**Remove Break Point**) from External Signal menu, the cursor change to Set Brk (Del Brk) then click the plug where you want set (remove) the break. To notify the inserted (removed) break the plug changes according to the figures in below "Cursor and Plug Summary" section.

The command **Remove all Break Point** in External Signal menu will remove the break points in whole project.

To know which and where the breakpoints are, use the **Cable List** command in UserList menu. The set breakpoint will be marked with the corresponding break type flags.

There are three different type of breakpoint:

1) BrkP – Pauses the running program
2) BrkW – Write on file
3) BrkI – Send an interrupt to Daq program

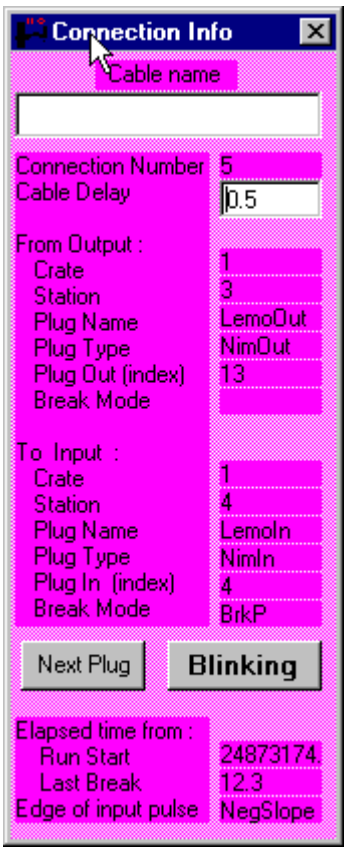
The user can set as many break points as he likes and the breakpoint types can be mixed but each plug can hold only one type. The assigned type is according to selected one in Tools/Options table, the selection will affect all the next breakpoint settings.

BrkP - break and pause

When a change on the voltage level occurs the program pauses and an expanded Connection Info table is shown. On that, as well as the usual connection parameters, the elapsed time from program start, elapsed time from the last break and the slope of the input pulse is shown too.

Using this type of breaks the user can have the continuous control of simulation and he can trace, step by step, the flow of signals and compute the relative timing from different parts of trigger system.

To resumes the program push the Run/Cont command button on TASS main window.



[1] In TASS, we call "event" any change in the level of signal on NIM o ECL plug.

BrkW – break and write to file

When a break occurs the following relevant variables will be written to break file and the simulation continue to run.

By default the break file name is "ProjectName".brk. The user can change the default name selecting a new one in Tools/Options table. The .brk extension will be add automatically.

Variables written on break file:

"-3,BrkIsr" = break identifier
nBrk = consecutive break number
PlugName = identifier name of the plug (LemoIn, EclIn etc.)
BrkTime = time elapsed since the beginning of run simulation
Crate = number of crate to which nMod belongs
Station = number of crate station where nMod sits
nMod = number of modules as reported in Title bar (blue bar on top of each modules)
Index = index identifying the plug inside nMod
Value = value of the voltage on the plug measured at BrkTime
BrkEdge = slop of the signal: Leading, Trailing, NoSlop (NoSlop meaning no change in voltage signal)

BrkI – break and send interrupt to Daq program (*)

When a break occurs an interrupt is sent to the **Daq.BrkIsr** routine supplied from Daq program and the simulation continue to run.

The parameters passed to Daq.BrkIsr are the same as in previous case, that is :

"-3,BrkIsr" = break identifier
nBrk = consucutive break number
PlugName = identifier name of the plug (LemoIn, EclIn etc.)
BrkTime = time elapsed since the beginning of run simulation
Crate = number of crate to which nMod belongs
Station = number of crate station where nMod sits
nMod = number of modules as reported in Title bar (blue bar on top of each modules)
Index = index identifying the plug inside nMod
Value = value of the voltage on the plug measured at BrkTime
BrkEdge = slop of the signal: Leading, Trailing, NoSlop (NoSlop NoSlop meaning no change in voltage signal)

This is the most powerful type of breakpoint, indeed the user, in the external Daq program, can selectively analyse the forwarded parameters and take the suitable actions. For instance, he can histogram the delay time between two critical points or to count how many hits arrive on a predefined input and so on.

(*) Concerning the Daq program and interrupt handling, see later "**Remote Daq**" section

Cursor and plug summary

Make connection (first plug)

Screwdriver-cursor, used to turn the setting screw on modules

Make connection (pending plug)

Hand cursor, general purpose cursor used to handle components on module

Delete cursor, to remove a connected cable

CTRL+Z To reset the cursor to standard cursor

Show cursor, to show the connected cables

Standard Lemo plug, not connected

Standard Lemo plug, connected (gray color)

Standard Lemo 50 Ohm terminator

Info cursor, to list cable connection properties

Standard Lemo plug, connected to data file (yellow color), see later.

Breakpoint cursor, to set and remove hardware breakpoint

Standard Lemo plug, connected to break point (light blue color)see later

Panel Picture	ToolBox picture	Picture folder (in ComponentBMP/)	Source code file (in ComponentSource/)
		Ecl_17x7/ Ecl_Hor Ecl_17x7/ Ecl_Vert	Ecl.ctl (Standard not connected)
	Ecl_Vert_Busy Ecl_Hor_Busy		Gray color (Busy)
	Ecl_Vert_Busy (invert) Ecl_Hor_Busy (invert)		Gray color (Busy)
	Ecl_Vert_File Ecl_Hor_File		yellow color (input data file)
	Ecl_Vert_File (invert) Ecl_Hor_File (invert)		yellow color (input data file)
	Ecl_Vert_Break Ecl_Hor_Break		light blue color (breakpoint)
	Ecl_Vert_Break (invert) Ecl_Hor_Break (invert)		light blue color (breakpoint)

Cursor's picture changes according to the selected function or selected component. In particular the screwdriver picture appears when the cursor is over a screw for setting of specific behaviour (threshold, width, delay etc.), the hand picture when cursor is over pushable/moveble components (button, switch, knob etc.).

5. Remote Tcp/Ip communication

Starting since Vers.4.0 has been introduced the new feature allowing TASS to be connect on the Internet network via Tcp/Ip protocol. This capability opens a new class of possible use of TASS system, that is:

- The full integration with the real Daq program can be easily achieved by the exchange of Camac and/or Vme command via Tcp/Ip communication (see next “Remote Daq” section).

- TASS can accept data issued from data file representing external electrical signals and send data to output file. This feature enable TASS to fill the gap between the existing packages for physics simulation (i.e. Geant) and the analysis program (i.e. Root) Fig.1, (see next "External Signal Data File " section).

- TASS can accept input data representing external electrical signals and send output data via a Tcp/Ip internet connection. This new feature promotes TASS to become not only a simulator but also a very powerful monitor of hardware trigger system (see next “Remote Hardware Monitor” section).

Overview of networking

Now we have to discuss a bit on the "Tcp/Ip Network" that we are going to use.
As well known two computer can be connected in different way depending from where they are sit.

- Word Wide Web

In this case all the structure of the web has to be available, in fact the computers are behind some Internet Service Provider (ISP) and all the work of "Network Address Translation" (NAT) is done by the same ISP.
The present version of TASS can't handle this feature, but TASS can carry out its full functionality if a <u>fix Ip number</u> is provided.
<u>Then if you foreseen to connect at remote computer in www you have to provide a fix Ip address for computer running TASS.</u>

- Local Area Network

This is the most frequent case in the real experiments.
In this case the computers (server and client) are sit on the same local internet subnet, then there is no need for the TASS computer to communicate to the external world. Indeed the communication between TASS and remote computers is handled simply by the router connecting both.
Consider that the router task can be done also by a cellular smart phone prepared to work as "personal hotspot", also without SIM telephone card.

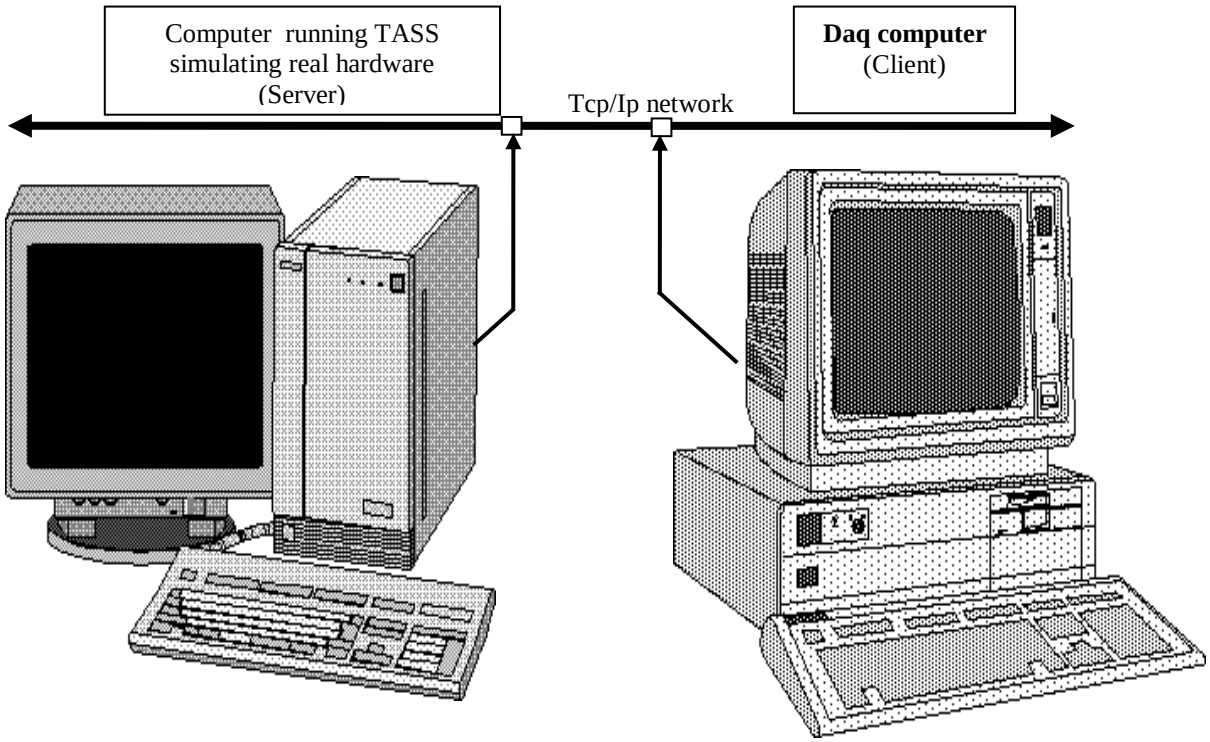
In any case to know the Ip number assigned from the router to the TASS computer you can open the menu "Help/About TASS" and watch the IP number in blue character, it should have the form 192.168.XXX.XXX where the "X" part depend from network connection.

6. Remote Daq

Camac and Vme are the most used standards for data acquisition and control in high energy physics experiments. As in the real world, where the Daq program steers the flow of commands to the hardware, as well in TASS the Daq program handles the virtual hardware. In both cases(*), the Esone/Vme library routines are the interface between Daq program and hardware modules.

As already mentioned the user can develop the real Daq program, using the final computer and the preferred operating system and programming language, in an environment where the real hardware is replaced by its TASS simulation and the cable connection to the Daq computer is replaced by the exchange of Camac/Vme commands via Tcp/Ip network communication (see following figure). This philosophy has important consequences:

- User can choose the best integration between hardware and software without any involvement of real hardware.
- Different parts of Daq and trigger programs can be developed simultaneously among more users even from worldwide spread sites.
- Comparison among different architectures can be done based on software simulation.
- None waste time for delivery delay of unavailable hardware.
- The documentation and the history of design process it's a built-in feature of TASS.
- Training of new users and/or updating of existing set up can be done without any involvement of real hardware.



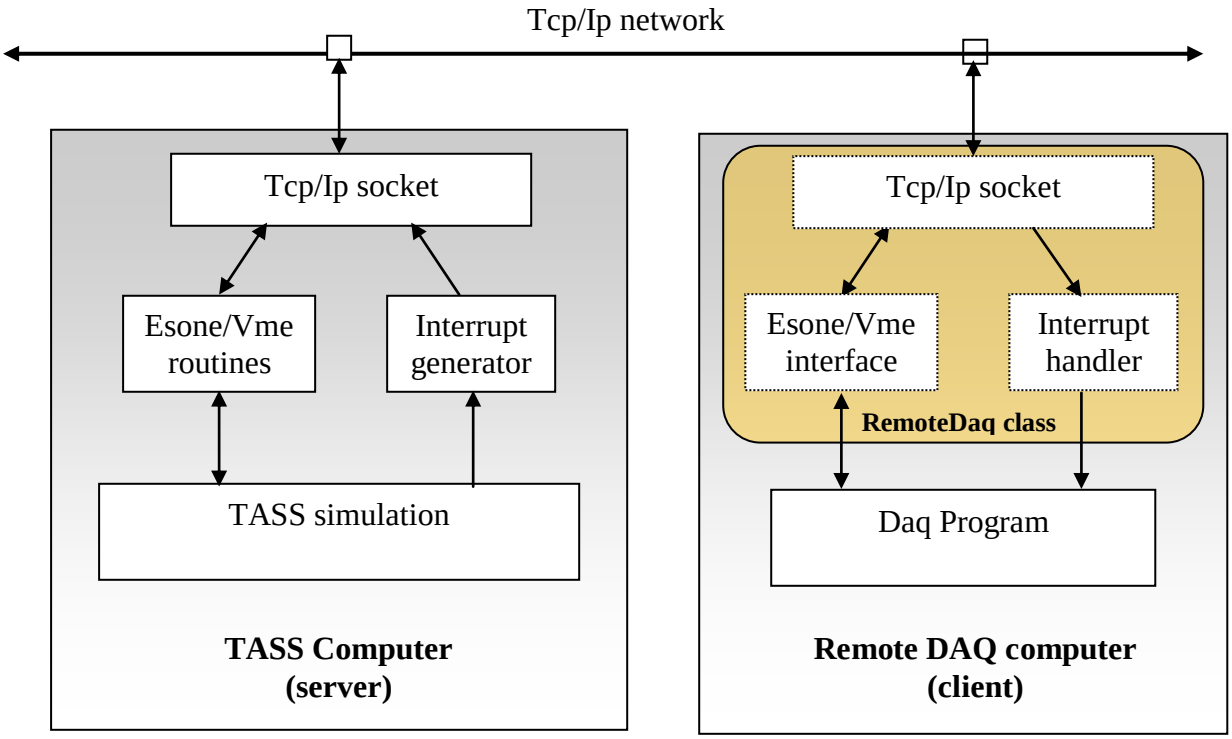
(*)In real world the standard Esone library provides routines to handle only the Camac modules as defined by the EUR 4100 and EUR 4600 recommendations. In TASS the routines supporting Vme interaction have been grouped together with the Esone routines in the EsoneVme class.

To achieve the above goals, TASS package supplies the software RemoteDaq class(*) providing the interface between the external Daq programs (user responsibility) and the TASS environment. *At present, supports for C++ and Java (in form of class), LabView (in form of VI) and Visual Basic (in form of .OCX control) are provided.*

The supplied RemoteDaq class (in the box on the right side figure) allows writing the Daq Camac/Vme calls in the same way as in the real world making the translation process, between the external Daq environment and the TASS system, fully transparent from user's point of view. Actually the user refers Camac and Vme hardware registers calling the standard Esone and standard Vme routines. The RemoteDaq class takes cure to translate the commands in suitable formatted messages and to send them over the network to TASS simulation computer.

Interrupts produced by TASS are treated in similar way and they provide events (in term of Visual Basic 'event' or 'call back procedure' for other languages) that triggers suitable procedures in user Daq program (refer to "**RemoteDaq class for communication protocol**" section).

The following figure summarizes the work done by the RemoteDaq class.



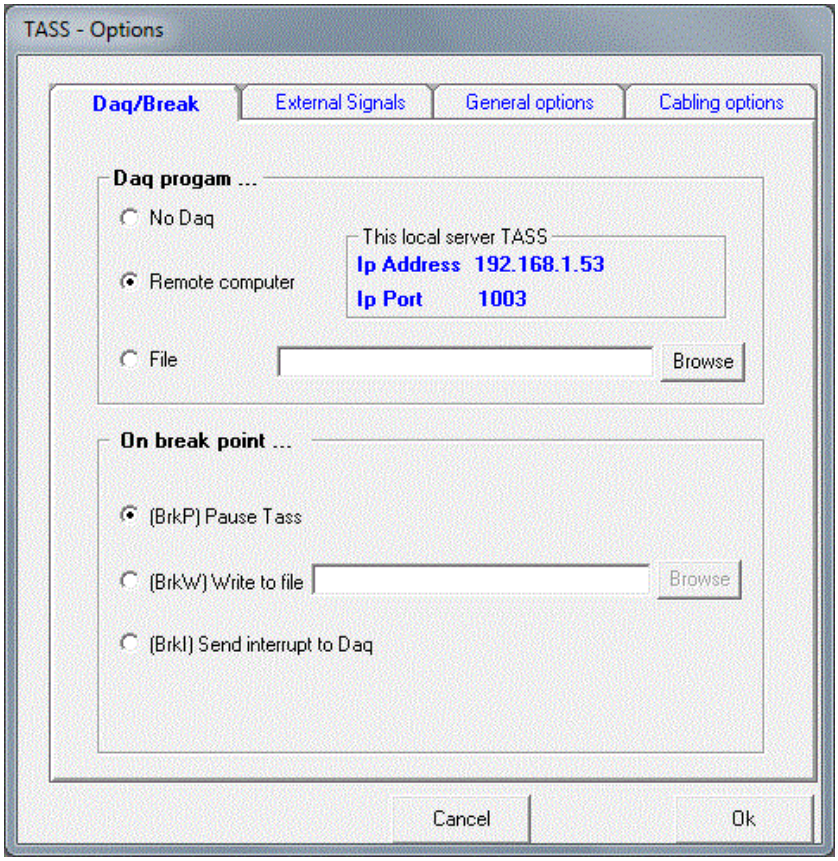
To handle the source of commands, open the **Tools/Option...** menu, then in **Daq/Break** tab in **Daq program** frame, you choose (see following figure)

- No Daq
  If none Daq program is foreseen (i.e only Nim modules used)

(*) The **RemoteDaq Class** is issued as separate package please download it from usual site www.top1.it/tass

- Remote computer
  If the Daq program is resident on different client computer (by default the port 1003 is reserved for Tcp/Ip communication). The Ip Address of TASS computer is shown, too(*).

- File
  If the Daq program (in .dll or .exe format) is resident on the same TASS computer



Of course the RemoteDaq class is depending from the operating system and language environment and few small differences can arise between them, nevertheless the following general features can be drawn.

- TASS always works as server computer and it reserves the port number **1003** for communication with external Daq program (see following "Overview of networking"). The server local Ip address is available on form open by **Help/About TASS** menu.
- To handle the RemoteDaq communications, select in TASS main window the **Tools/Options** menu and in the **Daq/Break** tab check the **Remote Computer** check box
- Always TASS simulation has to be started by the **Run/Cont** button or menu.

(*)Alternately, to know the IP number of the TASS server computer you can use the "**Help/About TASS...**" menù

- The set of Camac/Vme commands handled by RemoteDaq class reproduces the Esone and Vme library routines (see **RemoteDaq class for communication protocol** section).
- The commands and interrupts are based on the exchange of ASCII message strings. For each Camac/Vme command sent to TASS an acknowledge message is generated upon the command has been executed. This ensures the synchronization between server and client tasks.
- No acknowledge has to be produced by the Daq program in answer to any interrupts.

RemoteDaq Programming Model

Here we report some main considerations that we have to take in account when programmable modules (Camac and Vme) are used. In the following examples we use the Visual Basic language and we suppose to use the **TassRemoteDaqCntr** control (located on Form1 form) that formally handles all the EsoneVme and the network communication procedures, see later. Analog considerations can be done if other languages are used, infact the supplied RemoteDaq classes solve any usual needs.

Camac and Vme register references (only for Visual Basic)

The Camac and Vme standards foresee a slightly different way to address the internal registers of their modules.

In the Camac case the external hardware is mapped as external computer I/O, while in the Vme case it is mapped as external computer memory. From this follows that in Camac we have to define a variable holding a reference to the address module register by the function CDREG and then execute operation on that by the foreseen function (CSSA in following example).

```
Dim DaqCntr  as Object                          ' define DaqCntr  variable as Object
Dim Adc0 as Variant                             ' define a variable to hold the reference
…….
Set DaqCntr = Form1.TassRemoteDaqCntr           ' take reference to TassRemoteDaqCntr control

DaqCntr.CDREG Adc0, br, cr, AdcStation, SubAdd0  ' take the reference Adc0 to the
                                                 ' external  address  register defined by
                                                 ' branch=br, crate=cr, station= AdcStation
                                                 ' and subaddress= SubAdd0

DaqCntr.CSSA  16, Adc0, Data, q                  ' write (func=16) on the register defined
                                                 ' by Adc0 the data=Data and return q status

DaqCntr.CSSA  0, Adc0, Data, q                   ' read (func=0) from the register defined
                                                 ' by Adc0 and  return Data and q values.
```

In Vme case, being the module register allocated in memory address space, the real Daq program can refer it by simpler read/write instructions, i.e:

```
rData = *Reg          ' read data from Reg
*Reg =  wData         ' write data into Reg
```

Where the *Reg is a pointer variable pointing to the effective hardware memory address.

In the TASS environment we can't use the same syntax because the *Reg can't represent the register locations of a simulated module. To make the Daq code, as much as possible similar to the code that user would write in real world, TASS provides two property procedures solving this aspect (this is valid only in Visual Basic world):

```
Private Property Get Reg(RegAdd As Long) As Long
    DaqCntr.VmeRW  RegAdd, 0, Reg                    'Function 0 = read
End Property

Private Property Let Reg(RegAdd As Long, ByVal NewValue As Long)
    DaqCntr.VmeRW  RegAdd, 16, NewValue              'Fnction 16 = write
End Property
```

Now, as similar way as previous example, we can refer to Reg address, that is:

```
rData = Reg(RegAdd)          ' read  rData from  register locate at address RegAdd
Reg(RegAdd)= wData           ' write wData  into register locate at address RegAdd
```

Indeed all the link between register value and its defined hardware location is done by the above property procedures and by the DaqCntr.VmeRW called routine, see example code in \Sample\TcpIp_Protocol\EEE\EEE_Client_Daq\ EEE_Client_Daq.vbp.

Real world vs. virtual world

As well known, in real world usually there are two way in which the Daq program interacts with hardware :

- Interrupt handling
- Polling technique

In the first case the Daq program is, most of time, either idle or doing something else (i.e. histogram). Only when the hardware requests attention (interrupt) then the Daq program executes the foreseen routines to answer to those interrupt. This is known as asynchronous process.

In the polling technique, vice verse, the Daq program is in a loop always checking a flag used as semaphore. When the hardware demands attention, it sets the flag to notify to Daq program an event occurred. Only then, the Daq program exits from loop and executes the foreseen routines to answer to that event. This is known as synchronous process.

In real world, therefore, there are two different environments: the hardware and the Daq software each one doing its own job and the link between them is either the interrupt or the semaphore.

The interrupt method, interacting very closely with operating system, has a fast response time but the programmer has take cure how to code, especially if concurrent programs are running. For these reasons it is usually used in large or long term set up.

The polling method, being simpler to code and to maintain, is used in small or temporary setup.

TASS can handles both techniques. In conclusion TASS simulates very well the real world, in fact :

a) Using interrupt, the Daq program responds very fast to the events. The write of interrupt service routine is the price the programmer has to pay.
b) Using polling, the Daq program responds slower but the Daq itself is simpler to write and to maintain.

The taste and needs of the user will determine which philosophy to choose.

RemoteDaq class for communication protocol

As already mentioned the TASS package provides the RemoteDaq class for most of software environments used in the nuclear and particles physic field. In this section we'll give a description of the communication protocol to perform Camac and Vme commands between TASS and the remote Daq system. User can find the source code of RemoteDaq class for each supported language in the corresponding folder supplied by RemoteDaqClass installation package. User can freely modify that code or use it as guide to develop class for languages and/or operating systems not supported in the present version of TASS system.

No matter which language or operating system you are using, always the RemoteDaq class exposes general methods to provide an interface versus the Camac and Vme standard commands and, moreover, there are some few methods used to handle the network communication.

In the following list we report the callable methods provided by the Visual Basic control **TassRemoteDaqCntr.ocx**(*). For any other language the user can find similar methods proper of the used language. Help files can be found in the corresponding folders.

Methods related to network communication

OpenConnection (sRemoteHost As String, [lngRemotePort As Long = 1003])

Open a Tcp/Ip connection on remote host sRemoteHost using the default port 1003. sRemoteHost accept both format: Ip number (i.e. 141.108.7.32) and Ip address (i.e. tass.cern.ch). In case the server and client computer are the same, then sRemoteHost = LocalHost or sRemoteHost = 127.0.0.1 (See also above "**Overview of networking"** to know the IP number assigned to TASS computer).

(*) List of code for the **TassRemoteDaqCntr.ocx** control is available dowloading the **RemoteDaqClass.zip** package from the **www.top1.it/tass** web site.

CloseConnection

Close the existing Daq communication socket connection.

CommStatus **Read only Property**

Return the status of Daq communication socket:

0 = sckClosed
1 = sckOpen
2 = sckListening
3 = sckConnectionPending
4 = sckResolvingHost
5 = sckHostResolved
6 = sckConnecting
7 = sckConnected
8 = sckClosing
9 = sckError

Methods for Camac and Vme control

********** VME handling functions **********

int getReg(int regAdd)

return un integer value read from **regAdd** register

void setReg(int regAdd, int data)

set the register **regAdd** with the integer **data** value

********** CAMAC handling functions **********

For each of following methods TASS provides a test on Camac dataway line **X** to check if the addressed module is present, if it is not an error is reported.
In **Tools/Options/GeneralOption** tab you can disable this feature.

CDREG (ExtRef As String, Br As Long, Cr As Long, St As Long, SubAdd As Long)

Defines the Ext physical address corresponding to a given Camac unit.
ExtRef ← The encoded physical address
Br → Branch number of Camac Crate Controller
Cr → Crate Controller number
St → Camac module station number
SubAdd → Camac subaddress module's registers

CSSA (ByVal Func As Long, ExtRef As String, Data As Long, QResp As Boolean)

Read/Write Camac register in single word (24 bit)
Func → Camac function
ExtRef → Camac physical address as defined in CDREG function
Data ←→ Camac read/write data (24 bit)
QResp ← Q response of the last Camac operation. Q=true if Camac Q=1

CCCC (ExtRef As String)

Execute C Camac function
ExtRef → Camac physical address as defined in CDREG function

CCCZ (ExtRef As String)

Execute Z Camac function
ExtRef → Camac physical address as defined in CDREG function

CCCD (ExtRef As String, DFlag As Long)

Set or clear Demand line on Crate Controller
ExtRef → Camac physical address as defined in CDREG function
DFlag → Action flag. If DFlag=0 clear else set Demand Crate

CCCI (ExtRef As String, IFlag As Long)

Set or clear Inhibit line on Crate Controller
ExtRef → Camac physical address as defined in CDREG function
IFlag → Action flag. If IFlag=0 clear else set Inibit Crate Controll

CCINIT (ExtRef As String)

Initialize Crate Controller clearing Inhibit and setting Demand lines.
ExtRef → Camac physical address as defined in CDREG function.

Interrupt from TASS

TASS can produces simulated interrupts (LamIsr)(*) in the same way as the real hardware does. Moreover TASS can request attention to the Daq program for Daq initialization (DaqInit), in occasion of a break point (BrkIsr) or in order to send general message to Daq (GeneralMessage).
All this requests (generally called *interrupts*) will be forwarded to the Daq program issuing a proper message via the Tcp/Ip connection. The message takes the following format

intFlag, parameter1, parameter2, …

where the parameters list is comma separated.

intFlag	parameters	meaning
-1	DaqInit	Daq initialization request by TASS. Usually this interrupt is sent by TASS when the user chooses the ResetRun menu
-2	LamIsr	Standard interrupt from TASS. The same as in real hardware
-3	BrkIsr**,** nBreak, PlugName, BrkTime, Crate, Station, nMod, Index,Value, brkEdge	Usually this interrupt is sent by TASS when a break point is actived on any plug (see BreakPoint on Cabling section)
-4	GeneralMessage,MessageTitle, MessageText	General message from TASS server. Usually this interrupt is very useful when debuging code of a new module under development.

For their nature the interrupts are asynchronous respect the normal flow of the Daq program, that means that they have not an acknowledge to any Camac/Vme command.
See the Libraries section for description of Interrupt call.

About the BrkIsr interrupt the parameters have the following mean:

nBreak = number of break enumerator
PlugName = identifier name of the plug (LemoIn, EclIn etc.)
BrkTime = time elapsed since the beginning of program
Crate = number of crate to which nMod belongs
Station = number of crate station where nMod sits
nMod = number of modules as reported in Title bar (blue bar on top of each modules)
Index = index identifying the plug inside nMod (use the **Connection Info** tool to know the plug index number)
Value = value of the voltage on the plug measured at BrkTime
BrkEdge = slop of the signal: Leading, Trailing, NoSlop (NoSlop meaning no change in voltage signal)

(*) LamIsr is the acronym for Look At Me Interrupt Service Routine, the standard name used in Daq language

Example for Visual Basic RemoteDaq.

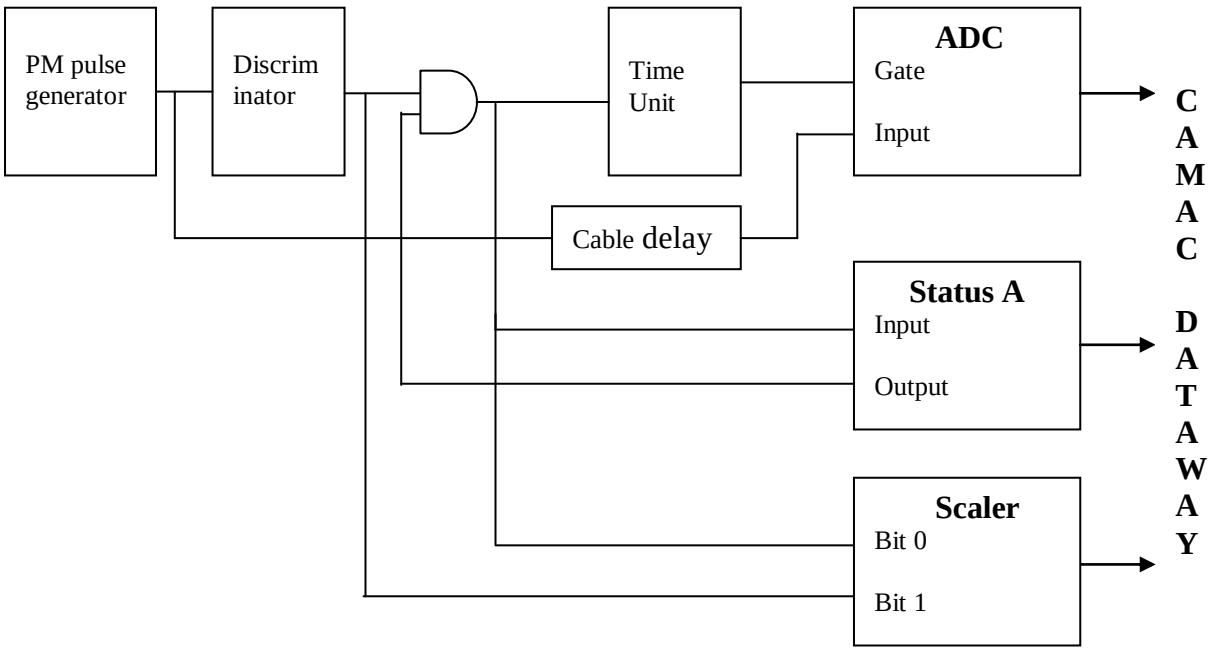
Here we report the fundamental steps to implement the communication protocol between the TASS and the remote computer running the Daq program.
We'll take as example the RemoteDaq_TestAdc_Lam program already used in several occasions.

For this purpose will use the Visual Basic language and the **TassRemoteDaqCntr** Control that has been already installed by the TASS installation process.

If user want use a different language he has to install on Daq computer the TassRemoteDaq class compatible with the operating system and language in use (refer to "**To install TASS**" section)

The RemoteDaq_TestAdc

In the following example we'll use, as usual, the RemoteDaq_TestAdc_Lam.trg (see Fig.2 in the "Introduction to TASS" section) small setup reproduced in the following block diagram and we'll use Visual Basic program languages to write the simple test Daq program.



The Sample package, installed together Tass, provides two Daq programs both located into Sample\TcpIp_Protocol\RemoteDaq_Adc folder: **RemoteDaq_TestAdc_Lam.exe** and **RemoteDaq_TestAdc_Polling.exe** using respectively the interrupt handling and polling technique.

User can use these programs without any external support; nevertheless, if he wants to do any changes to the programs, he needs the Visual Basic 6 compiler available

To try this example

- **on TASS computer** (server)
  - start TASS
  - load the **RemoteDaq_TestAdc_Lam.trg** trigger setup file
  - Start the simulation by the Run/Cont button
  - Move switch on NimWave generator in Run position

- **On Daq computer** (Client)
  - Start the **RemoteDaq_TestAdc_Lam.exe** program
  - In the appearing input box insert the name or Tcp/Ip number address of the remote computer runnig TASS (open the **Help/About Tass** menu to know the IP number assigned to TASS computer). The Daq program start to collect and histograms the acquired data.

If user want modify any part of Daq program:

- **On Daq computer** (Client – with Visual Basic 6 installed)
  - Start the Visual Basic.
  - In the Windows Explorer navigate and double click on **RemoteDaq_TestAdc_Lam.vbp** in \Sample\TcpIp_Protocol\RemoteDaq_Adc\RemoteDaq_Adc_Lam folder.
  - Visual Basic starts, change any part of code then run the program.

To try the polling technique replace the RemoteDaq_TestAdc_Lam.exe in the previous step with the file **RemoteDaq_TestAdc_polling.exe.**

How it works

The Pulse Generator produces simulated photomultiplier signals, the Discriminator and the Time Unit produce the gate for ADC. The StatusA and the Coincidence logic allow only one event at once be sent to the ADC. The Scaler counts the number of event treated. The length of cables is chosen to match the proper delays. The interrupt will be generated from ADC at end of conversion time.

Refer to the data sheets of constructors to know how each module works.

Upon the client program starts the run, a request for connection will be send to TASS computer and, if any thing is ok, a connection will be established. The TASS simulation on the server computer, for each suitable triggered event, will produce an interrupt that will be sent to the client via network(*).

On the client side, the reception of interrupt message will activate in Form1.frm the corresponding **TassRemoteDaqCntr_LamIsr** procedure that, in turn via a call to InterruptService procedure, requires the read of Camac Adc and Scaler modules then the resulting values will be histogrammed and written on the Test.daq file.

Visual Basic allows user to trace the Daq program stepping instruction by instruction and using whole standard debug procedure supplied by the environment.

In the following programs list what concern the Camac, Vme and histogram packages please refer to the proper **Libraries** section.

Note: you can found an equivalent Daq program written in Java in TassProject\RemoteDaqClass\RemoteDaqJava\RemoteDaqJava_Lam pakage

(*) The interrupt is a special message, see later **Communication protocol for RemoteDaq class.**

Client RemoteDaq_TestAdc_Lam.vbp Program Code

========== partial Code in Form1.frm ==========

```
Private Sub TassRemoteDaqCntr_ClosingConnection()
  MsgBox "Connection closed by server", vbOKOnly, App.Title
End Sub

Private Sub TassRemoteDaqCntr_error(ByVal Number As Integer, Description As String)
  MsgBox Number & ", " & Description, vbOKOnly, App.Title
  If Number = 21 Then End
End Sub

Private Sub TassRemoteDaqCntr_LamIsr()
  Call InterruptService
End Sub
```

========== Code in Module1.bas ==========

```
'Declare Objects
Public DaqCntr As Object

'Declare variables used in CDREG references
Public Adc0 As String              'reference at Adc channel 0
Public Adc1 As String              'reference at Adc channel 1
Public Scaler0 As String           'reference at Scaler channel 0
Public Scaler1 As String           'reference at Scaler channel 1
Public StatA0 As String            'reference at StatusA channel 1
Public StatAMask As String         'reference at StatusA Mask register
'......                    'any other references go here

'Declare any other global variables
Dim q As Boolean
Public DataLun As Long, TotEv As Long, NEv As Long
Public BlockNum As Long, Index As Long
Public DataMessage As String, DataValue As Single, DataTime As Currency

Public Timer0 As Single
'......                    'any other reference goes here
```

```
Public Sub Main()
Dim RemoteAddress As String
'Visual Basic starting procedure for client Daq program

  Set DaqCntr = Form1.TassRemoteDaqCntr                    'define reference Object for DaqCntr library

   RemoteAddress = InputBox("Insert the remote Tcp/Ip address." & vbCrLf & _
                   "It can be either the Tcp/Ip number or its address name", _
                   "Remote Daq", "top1.roma1.infn.it")

  DaqCntr.OpenConnection RemoteAddress
  Call Initialize
End Sub
```

```
Public Sub Initialize()
'This routine is used to initialize the Camac and program variables

Dim NameFile As String
Dim br As Long, cr As Long
Dim AdcStation  As Long, ScalerStation As Long, StatusAStation As Long
Dim SubAdd0 As Long, SubAdd1 As Long
Dim Mask As Long
Dim Data0 As Long, Data1 As Long
Dim CounterData0 As Long, CounterData1 As Long
'.....                  'any other variable definition goes here

  Close                                    'close all active open files
  NEv = 0                                  'current event
  TotEv = 50000                            'total events

  NameFile = CurDir & "\Test.daq"          'set file name where put read data

  DataLun = FreeFile                                    'get free Logical Unit
  Open NameFile For Output Access Write As DataLun      'open file


'initialize and book histograms
  Form1.cntrHBook(0).HBook " data0", 0, 400, 100          'book histos for Adc Ch 0
  Form1.cntrHBook(1).HBook " data1", 0, 200, 50           'book histos for Adc Ch 1
  Form1.cntrHBook(2).HBook " system time", 0, 10, 100     'book histos for system clock
  Form1.cntrHBook(3).HBook " dead time", 100, 150, 100    'book histos for dead time (microsec)

'initialize the Camac
  br = 3                                  'Branch number as defined in Layout form
  cr = 2                                  'Crate number as defined in Layout form
  AdcStation = 2                          'in this Station put LeCroy ADC_2249
  ScalerStation = 4                       'in this Station put CAEN Scaler C257
  StatusAStation = 6                      'in this Station put CAEN StatusA C236
  SubAdd0 = 0                             'sub address for Adc channel 0
  SubAdd1 = 1                             'sub address for Adc channel 1

  DaqCntr.CDREG Adc0, Br, Cr, AdcStation, SubAdd0     'define DaqCntr register Adc0
  DaqCntr.CDREG Adc1, Br, Cr, AdcStation, SubAdd1     'define DaqCntr register Adc1
  DaqCntr.CDREG Scaler0, Br, Cr, ScalerStation, 0     'define DaqCntr register for scaler Ch 0
  DaqCntr.CDREG Scaler1, Br, Cr, ScalerStation, 1     'define DaqCntr register for scaler Ch 1
  DaqCntr.CDREG StatA0, Br, Cr, StatusAStation, 0     'define ch 0 register for StausA

  DaqCntr.CCCZ Adc0                       'initialize Camac crate

  Mask = &H401                            'define mask = enable inp+Busy ch 0 (1)
  DaqCntr.CSSA 17, StatA0, Mask, 0        'write StatusA Mask Register

  DaqCntr.CSSA 9, Adc0, 0, 0              'reset Lam and Adc module
  DaqCntr.CSSA 10, StatA0, 0, 0           'reset StatusA /Busy output
  DaqCntr.CSSA 26, Adc0, 0, q             'enable Adc Lam

  Timer0 = Timer                          'up to date system time

End Sub
```

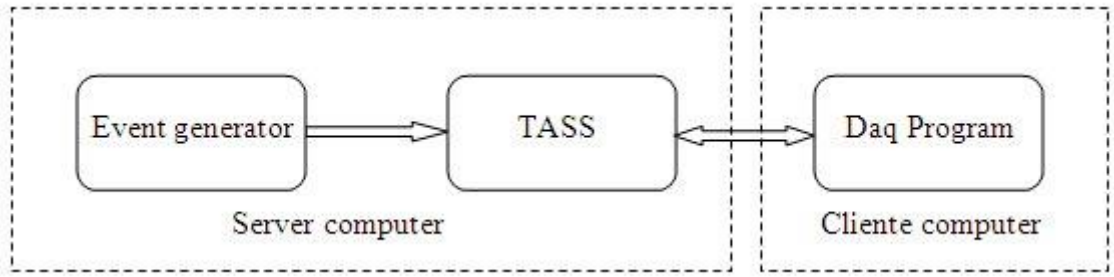
```
Public Sub InterruptService()
Dim Data0 As Long, Data1 As Long, CounterData As Long

'this routine (Interrupt Service Routine) handles any Lam from
'the Camac module

  DoEvents                              'leave time to OS

'  Beep                                 'sound to notify a new event

'execute camac  function
  DaqCntr.CSSA 2, Adc0, Data0, q        'read ADC ch 0
  DaqCntr.CSSA 2, Adc1, Data1, q        'read ADC ch 1 (only pedestal on this channel)
  DaqCntr.CSSA 0, Scaler0, CounterData, q   'read scaler ch 0

'=== write data to file
  Write #DataLun, NEv, " Adc0,Adc1,Scaler,Time  ",Data0, Data1, CounterData, Timer

'=== write data to window
  With Form1.TextBox
    If Len(.Text) > 10000 Then
      .Text = Mid(.Text, 5000)          'cut text in TextBox
    End If
    .Text = .Text & vbCrLf & _
          " Adc0,Adc1,Scaler     " & _
          Data0 & "    " & Data1 & "    " & CounterData
    .SelStart = Len(.Text)
  End With

'=== fill the histograms
  Form1.cntrHBook(0).HF1 CSng(Data0)        'histos Data0
  Form1.cntrHBook(1).HF1 CSng(Data1)        'histos Data1
  Form1.cntrHBook(2).HF1 CSng(Timer - Timer0)   'compute the esecution time per event (0.1 sec/bin)

  Timer0 = Timer

  Form1.cntrHBook(optHistoIndex).ShowHist   'refresh all visible histos

  NEv = NEv + 1                             'count the events
  If NEv = TotEv Then
    Close DataLun                           'close open file
    DaqCntr.CloseConnection
    End                     'close properly the link with TASS
  End If

'==== reset the hardware
  DaqCntr.CSSA 9, Adc0, 0, q                'reset Lam and Adc module
  DaqCntr.CSSA 10, StatA0, 0, q             'reset StatusA /Busy output

End Sub
```

Full simulation of Extreme Energy Events Detector and Daq

In this second example we'll show a full simulation of a real (small) experiment, please refer to http://www.top1.it/Tass/TassManual/EEE_InternalNote.pdf (see also https://eee.centrofermi.it) for any details. The EEE experiment aims to study the behavior and origin of very high energy cosmic rays using a telescope made by 3 MRPC detectors.

Our simulation consists in two parts (see figure):

- the server computer provides support for both tasks: the generator of physical events (the cosmic rays) and the TASS program.
- the client computer runs the data acquisition and analysis program (Daq).



Indeed, the event generator and the TASS program talk each other via network protocol also if, really, they are sitting on the same computer. The protocol used is just that described in "**External Tcp/Ip signal data**" in previous **"Remote Tcp/Ip Communication"** section.

Note that TASS program works as client respect to the event generator but as server respect to the Daq program.

The event generator produces muons with $\cos^2$ (Theta) distribution impact angle on the detector and provides strip's signals that will be collected and sent to TASS trigger simulation.
The Time Of Fligth (TOF) and the strip's delay for each hit is computed and sent to TASS input of Caen_V1190B module

The event generator produce also a FastOr signal (one for each plane) when the corresponding plane is hit. The FastOr signals are sent to FanIn-FanOut (mod.N454) front end Nim module and merged all together to produce a triple coincidence. A veto input, under computer control by the /OR output from StatusA modules (mod. V977), the coincidence handles the correct handshake (see also the equivalent **The RemoteDaq_TestAdc section**).

Each event will be processed as in the real hardware then, for events fulfilling the trigger request (all planes hit), an interrupt will be generated to alert the Daq program. Upon the Daq receive the interrupt, suitable Vme comands will be issued to collect data and the relevant analysis will be done to extract the cos(Theta)and Cos(Phi) distribution.

Server Side

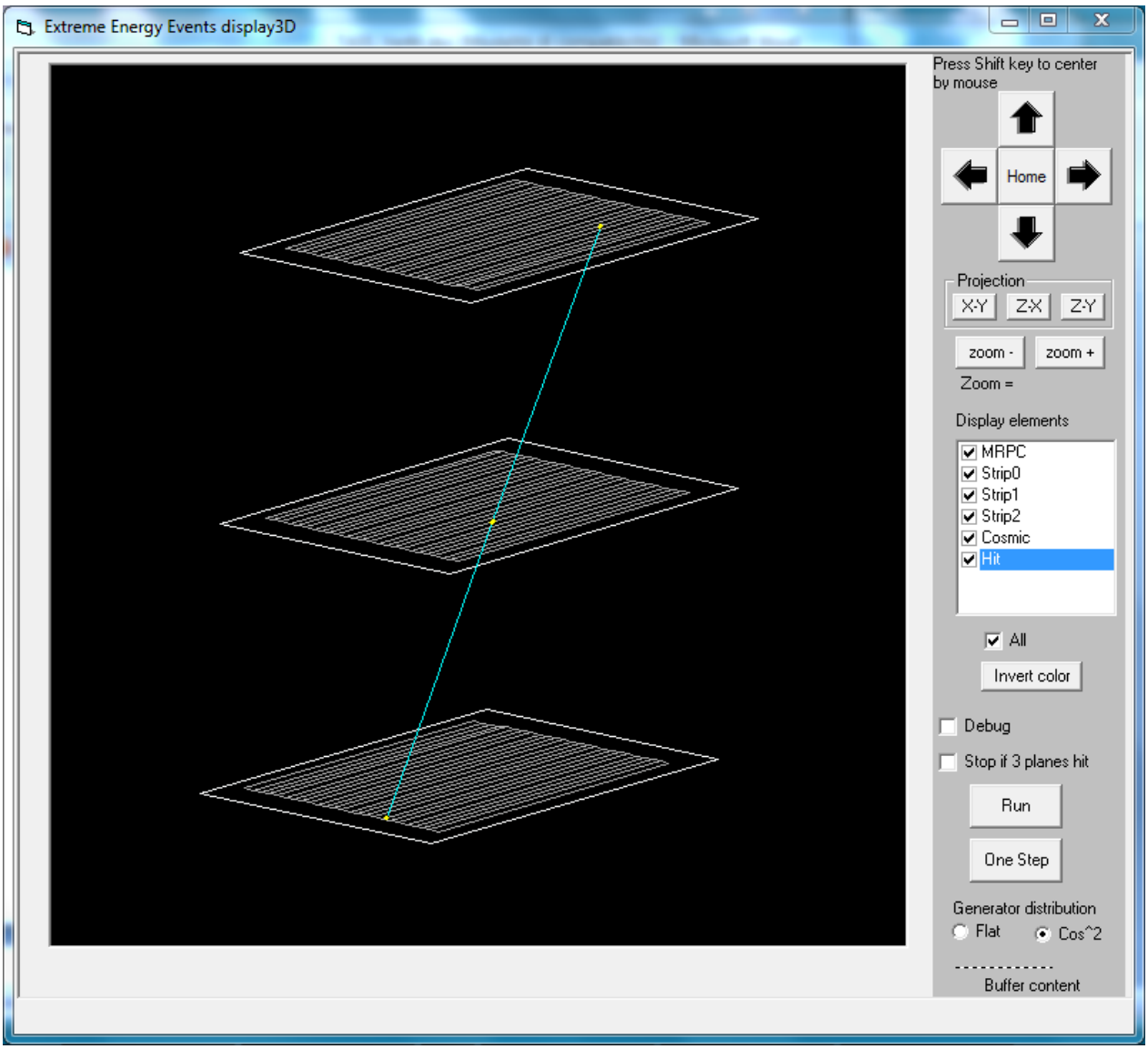
To try this example we need a server computer running Windows operating system and connected to the network (see the previous **Overview of networking** section to know the Ip number assigned to TASS computer), then proceed in the following step:

1. On the server computer install the TASS package (see “To install TASS” section).

2. In the **/Sample/TcpIp_Protocol/EEE/EEE_Server** folder run the **EEE_EventGenerator.exe** program.
3. Start the event generator by **Run** button. The trace of random cosmic are shown.

(As usually, if the Visual Basic system is present on the server, you can edit and modify any part of code (refer to Microsoft Visual Basic documentation), opening the **EEE_ EventGenerator.vbp.)**

Running the **EEE_EventGenerator.exe** program, you will get the following picture showing a schematic view of the EEE detector.



3D representation of EEE simulated events

The detector's graphical representation consists in two parts:

- The 3D display on the left, based on the cntr_Display3D.ocx. This is a general purpose 3D display that has been installed by TASS installation procedure and anyone can use it, refer to its manual in the EEE_Server folder.

- The command section, on the right side, where several cntr_Display3D.ocx features are handled.

Again on the server computer, start TASS and select the simulation trigger set up **/Sample/TcpIp_Protocol/EEE/EEE_Server/EEE.trg**, which sketch and picture are reproduced in the following pictures.

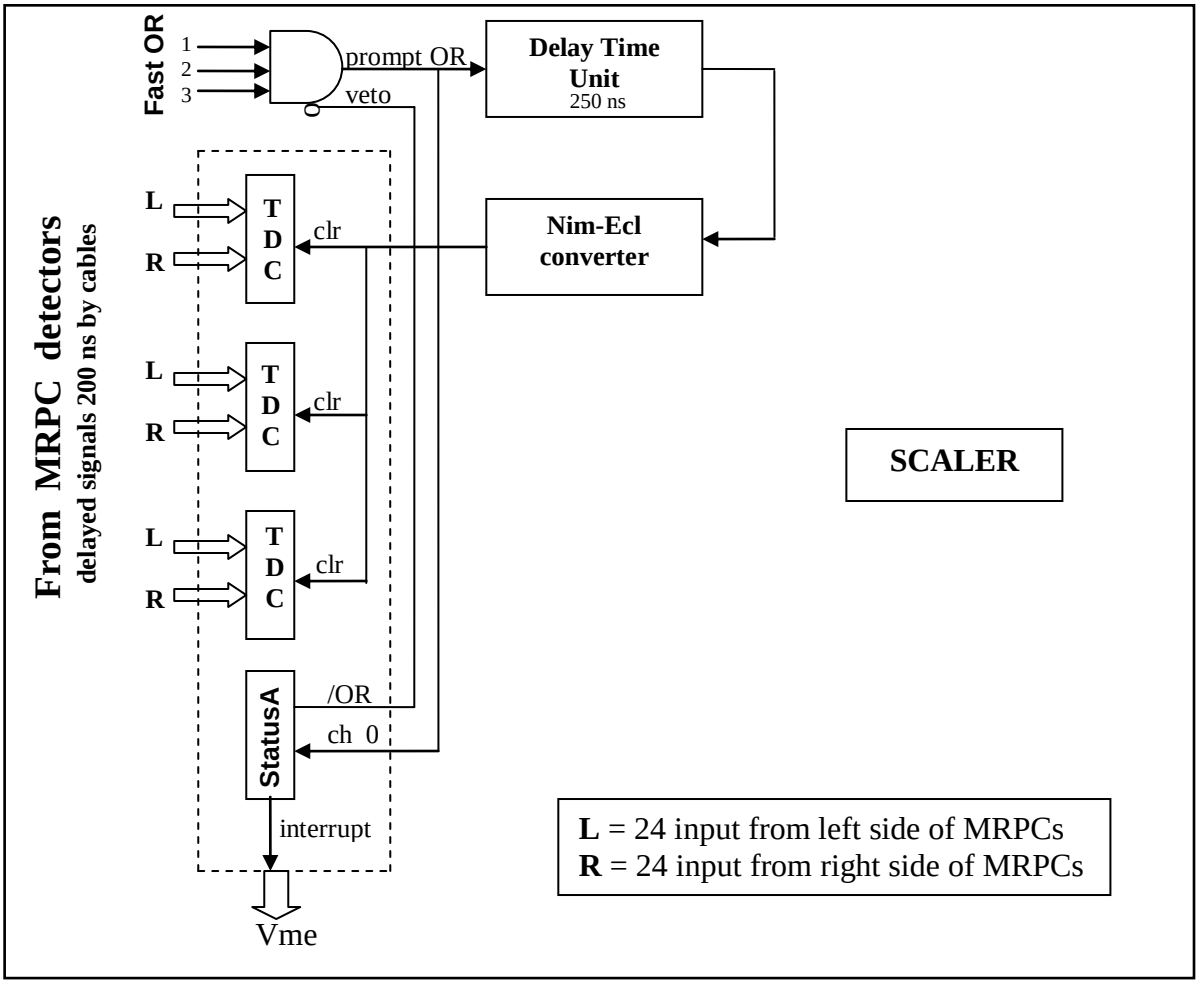
On TASS window select **Tool/Options**… menu and :

1. in the **External Signals** tab make sure, in the **input signals from...** frame, the **Remote computer** option box is selected, then write down the Tcp/Ip address **"127.0.0.1"** (this is the LocalHost address) and the default port numbers **1001** (*).

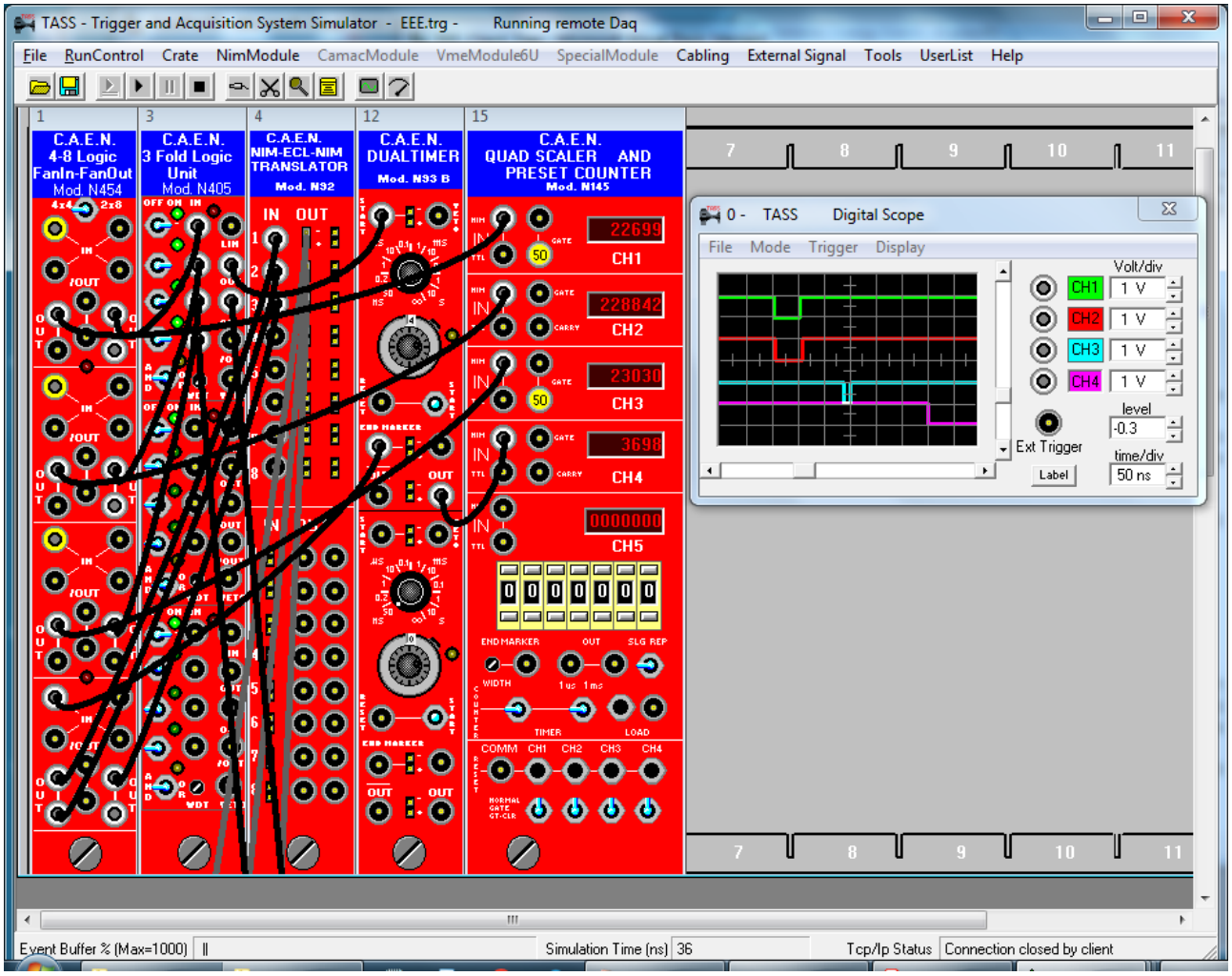
2. in **Daq/Break** tab make sure the **Remote computer** option is selected. Take note of Ip Address number in "This local server TASS" frame, then **Ok** button.

3. Start the simulation by **Run/Cont** button in TASS main window.

(*) Of course the events generator program can be on a different computer where TASS is running, in this case introduce in the address field the corresponding Tcp/Ip number.



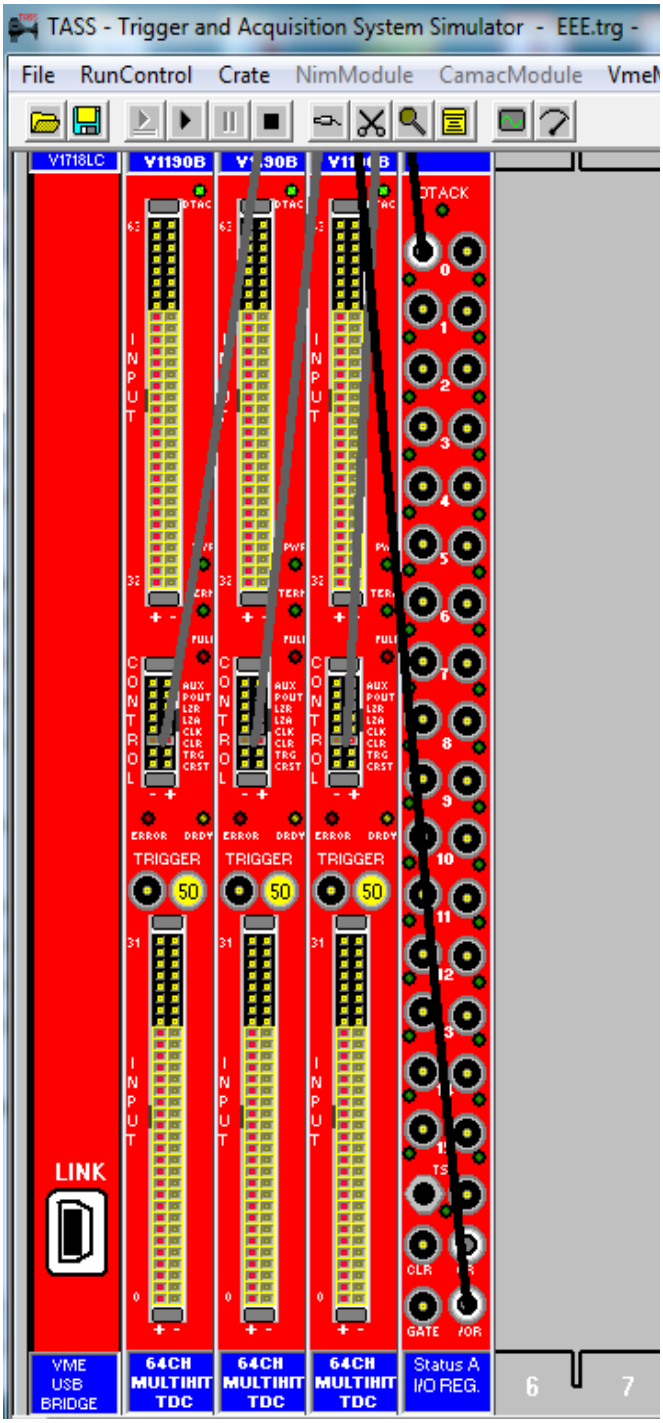
skim of acquisition system



Trigger system (Nim crate) of EEE.trg

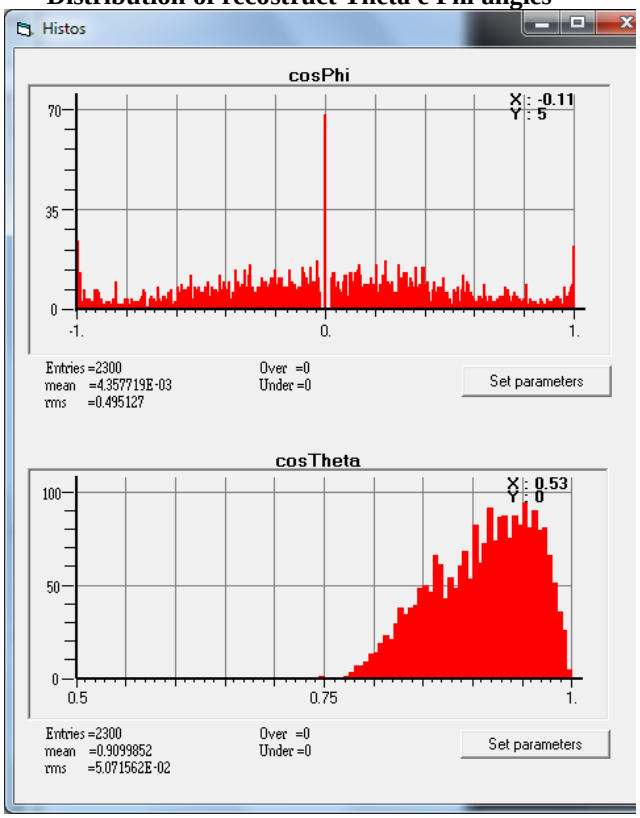
Used modules for full set up

TDC	**=**	**Caen V1190B**	**Scaler**	**=**	**Caen N145**
StatusA	**=**	**Caen V977**	**Nim-Ecl**	**=**	**Caen N92**
Coincidence	**=**	**Caen N405**	**Fan Out**	**=**	**Caen N454**
Time Unit	**=**	**Caen N93B**			



Trigger system (Vme crate) of EEE.trg

Distribution of recostruct Theta e Phi angles (*)



done by the client Daq program

The incoming **External Data Signals,** provided by the events generator, sent via local network will be associated to the corresponding TDC input as defined in the **EEE.trg** file. User can see the full association list by **UserList/SignalConnectionList** menu. To remember how to do this procedure refer to **How to assign an external Tcp/Ip input signal to input plug** next section.

User can see on the scope the flickering incoming signals of the FastOr output from the detector planes 1 and 2 and the number on the counter increasing.

(*) The peak at zero degree in cosPhi histogram is due to an artifact of muon direction reconstruction

Client Side

As well already said, the client Daq program can be written in any languages and under any operating system you like. Here we report the Visual Basic version, Java version follows.

We have to connect the Daq program running on the client computer, then run the **Sample/TcpIP_Protocol/EEE/EEE_Client_Daq/EEE_Client_Daq.ex**e

Upon the program starts, it asks for a Tcp/Ip connection to the server, type in the Ip Address you have took in note earlier in the server side setting (otherwise you can see the previous Overview of networking section to know the Ip number assigned to TASS computer), if everything is correct, a communication will be established (a sound warn you a connection has been established !).

For each triggered event in the server (events having all the planes hit) an interrupt is sent to client Daq program that will handle all the suitable commands to collect data from TASS simulation and to fill the histograms(*).

The Daq program (in Visual Basic language) is shown in following pages, the Java version follows. The most relevant code is inside the routines **Initialize** and **InterruptService** called respectively at begin of start up and any time the hardware interrupt signal occurs.

In the **Initialize** procedure the **Caen_V1190B TDC**s have been programmed to work in Continuous Storage Mode (see Caen_V1190B manual) that means the inputs work as start signal and the Clr as a stop signal.
The triple coincidence of the **Fast Or** signals provided by the front end electronics of each chamber, with suitable delay has been used to shape the **Clr** signal to TDC. The interrupt signals will be generated by the **StatusA** module (**Caen V977)**. Each interrupt will activate a call to the **InterruptService** procedure of the **EEE_Client_Daq.exe** program (see listing). In this procedure the Vme modules will be read out and the relevant parameters, Cos(phi) and Cos(Theta), computed and displayed on the histograms.

(*) If, on the client computer, you have installed the Visual Basic framework you can edit and recompile any source code of Daq program.
DoubleClick on **Sample/TcpIp_Protocol/EEE/EEE_Client_Daq/EEE_Client_daq.vbp**

Visual Basic Client EEE_Remote_Daq.vbp Program Code

========== Code in EEE_Daq.bas module ==========

```
Public Sub Main()
Dim RemoteAddress As String

'Visual Basic starting routine      DO NOT change or erase

  On Error GoTo errHandler

  Set DaqCntr = EEE_frmHist.TassRemoteDaqCntr

  RemoteAddress = InputBox("Insert the remote Tcp/Ip address." & vbCrLf & _
                  "It can be either the Tcp/Ip number or its address name", _
                  "Remote Daq EEE", "IpNumber of your server computer")

  DaqCntr.OpenConnection RemoteAddress              'request connection to server
  Call Initialize
  Exit Sub

errHandler:
  MsgBox Err.Source & ", error n. " & Err.Number & vbCrLf & Err.Description
  End
End Sub
```

```
Public Sub Initialize()
Dim br As Long, cr As Long, StatusAStation As Long
Dim Mask As Long

  DaqCntr.TassResetRun                    'reset hardware at idle state

  EEE_frmHist.cntrHBook1.HBook "cosPhi", -1, 1, 100
  EEE_frmHist.cntrHBook2.HBook "cosTheta", 0.5, 1.5, 100

  BaseAdd(1) = &H100000                   'base address for Caen_V1190B in stat 1
  BaseAdd(2) = &H200000                   'base address for Caen_V1190B in stat 2
  BaseAdd(3) = &H300000                   'base address for Caen_V1190B in stat 3
  BaseAdd(4) = &H400000                   'base address for Caen_V997 in stat 4

  Reg(BaseAdd(1) + &H100A) = 0            'set interrupt level V1190B in stat 1
  Reg(BaseAdd(1) + &H1022) = 3            'Almost Full Level   V1190B in stat 1
  Reg(BaseAdd(2) + &H100A) = 0            'set interrupt level V1190B in stat 2
  Reg(BaseAdd(2) + &H1022) = 3            'Almost Full Level   V1190B in stat 2
  Reg(BaseAdd(3) + &H100A) = 0            'set interrupt level V1190B in stat 3
  Reg(BaseAdd(3) + &H1022) = 3            'Almost Full Level   V1190B in stat 3

  Reg(BaseAdd(4) + &HE) = 3               'interrupt mask for V977 (all channel enabled)
  Reg(BaseAdd(4) + &H20) = 7              'interrupt for V977  enabled
  Reg(BaseAdd(4) + &HA) = &HFF            'set/reset the all output of V977
  Reg(BaseAdd(4) + &HA) = 0               'set/reset the all output of V977


  EEE_frmHist.Show
  EEE_frmHist.cntrHBook1.hParamVisible = True
  EEE_frmHist.cntrHBook2.hParamVisible = True
End Sub
```

```
Public Sub InterruptService()
Dim q As Boolean
Dim Chan As Integer, LChan As Integer, RChan As Integer
Dim RawData As Long, EventStored(1 To 3) As Integer
Dim Data As Single
Dim LTim As Single, RTim As Single
Dim LDist As Single, RDist As Single
Dim X(1 To 3) As Single, Y(1 To 3) As Single
Dim L00 As Single, L10 As Single
Dim cosPhi As Single, cosTh As Single

On Error GoTo errHandler

 nEv = nEv + 1
 EventStored(1) = Reg(BaseAdd(1) + &H1020)          'retrieve the accumulate hit number
 EventStored(2) = Reg(BaseAdd(2) + &H1020)
 EventStored(3) = Reg(BaseAdd(3) + &H1020)

'  Debug.Print "EventStored", EventStored(1), EventStored(2), EventStored(3)
 For j = 1 To 3
  evStor = EventStored(j)
  For i = 1 To evStor
   RawData = Reg(BaseAdd(j) + &H0)                  'row data from TDC
   Chan = (RawData And &H3F80000) \ 2 ^ 19          'mask and get the channel index
   Data = (RawData And &H7FFFF) / 10                'get time value

   If Chan <= 31 Then
     LTim = Data                          'left channel
   Else
     Chan = Chan - 32
     RTim = Data                          'right channel
   End If
  Next

  LDist = 0.5 * (LTim - RTim + strTim) / tSpeed      'longitudinal position (meter)
  RDist = 0.5 * (RTim - LTim + strTim) / tSpeed

  Debug.Print "  data  ", Chan, RawData, LTim,  RTim, LDist, RDist

  X(j) = (Chan * strDist - Xd)                       'transversal position (cm)
  Y(j) = LDist * 100 - Yd
 Next

'--compute Cos(phi) and  Cos(theta)
 L00 = Sqr((X(1) - X(3)) ^ 2 + (Y(1) - Y(3)) ^ 2)
 L10 = Sqr(L00 ^ 2 + 4 * Z0 ^ 2)
 cosPhi = (X(3) - X(1)) / L00
 cosTh = 2 * Z0 / L10

'  Debug.Print "cosPhi, cosTh ", cosPhi, cosTh
'  Debug.Print

 Call EEE_frmHist.cntrHBook1.HF1(cosPhi)            'histogram
 Call EEE_frmHist.cntrHBook2.HF1(cosTh)
 Call EEE_frmHist.cntrHBook1.ShowHist
 Call EEE_frmHist.cntrHBook2.ShowHist

 Reg(BaseAdd(4) + &H10) = 0                         'reset all channel interrupt but Not the Mask reg

 Exit Sub
```

```
errHandler:
'  MsgBox Err.Description, vbOKOnly
  Reg(BaseAdd(4) + &H10) = 0                'reset all channel interrupt but Not the Mask reg  Exit Sub
End Sub
```

========== Code in EEE_frmHist.frm module ==========

```
Private Sub TassRemoteDaqCntr_ClosingConnection()
   MsgBox "Connection closed from remote computer"
End Sub

Private Sub TassRemoteDaqCntr_LamIsr()
  Call InterruptService
End Sub
```

========== Code in EEE_DetecConst.bas module ==========

```
Public X0 As Single, Y0 As Single
Public X1 As Single, Y1 As Single
Public X2 As Single, Y2 As Single
Public L00 As Single, L10 As Single

Public Const W As Single = 50
Public Const L As Single = 100

Public Const Z0 As Single = 100          'z position of chamber
Public Const Z1 As Single = 0
Public Const Z2 As Single = -100

Public Const Xd  As Single = 42          'sensible area
Public Const Yd  As Single = 85
Public Const Zd  As Single = 0

Public Const strDist As Single = 3.5          'strip dist
Public Const strWdt As Single = 2.5           'strip width
Public Const strLng As Single = 170           'strip length
Public Const tSpeed As Single = 4.5           'strip signal speed 4.5 ns/m
Public Const cSpeed As Single = 3.3           'light speed 3.3 ns/m
Public Const strTim As Single = tSpeed * strLng / 100
```

Java Client EEE _Daq.java Program Code

The following code is a skeleton of Java version for client **EEE_Daq.java.**

Note: you can found the full Daq program (also the DaqPack.java and hbook.java package) written in Java in Tass\Sample\RemoteDaqClass\RemoteDaqJava\EEE_DaqJava package. Is user responsibility to implement the code inside the suitable IDE for Java environment (i.e. Eclips, Netbeans etc).

```
/*
  Skeleton example of Remote Data Acquisition program for
D:\ProgramFiles\VB98\TassProject\Sample\TcpIp_Protocol\EEE\EEE_Server\EEE.trg trigger
system
  Refer to www.caen.it/download/?filter=V1190b and www.caen.it/download/?filter=V977
  for manual of CAEN_V1190b and CAEN_V977 Vme modules

 Usage:
Prompt>  javac eee_client_daq/EEE_Daq.java
Prompt>  java  eee_client_daq/EEE_Daq
*/

package eee_client_daq;

        import java.io.*;
        import java.net.*;

        import javax.swing.JFrame;
        import java.awt.Color;                          //<<< add

        import org.opensourcephysics.display.*;
        import org.opensourcephysics.display.Dataset;
        import org.opensourcephysics.frames.PlotFrame;


public class EEE_Daq {

        static String TassMessage;
        static int  nEv;
        static int[]  BaseAdd={0,0,0,0,0};
        static int  BlockNum, Index;
        static double DataValue, DataTime;
        static double McTh, McPhi;
        static String DataMess;

//<<<< Tcp/Ip RemoteDaqAdd Computer, to be update, see     Help/About TASS menu and also
Remote Daq section

//write down the correct Ip number of  TASS computer
//      static String RemoteDaqAdd = "192.168.1.61";
        static String RemoteDaqAdd = "127.0.0.1";
```

```
        static DaqPack Daq;
        static HBook H;


         String comand;
         String message;
         String returnMessage;

public static void main(String[] args) throws Exception {
         Daq = new DaqPack();                    // instance Daq Package
         Daq.Connect(RemoteDaqAdd);

         H = new HBook();                        // instance HBook Package
        // H.hInit("Cos(theta)", 0.5, 1, 100);
         H.hInit("Cos(Phi)", -1, 1, 200);

         H.hColor("BLUE");
         H.hXYlabel("ADC channel", "Content");       // da capire perche' non funziona
         H.hType("BAR");

         BaseAdd[1] = 0x100000;                  // base address for Caen_V1190B in stat 2
         BaseAdd[2] = 0x200000;                  // base address for Caen_V1190B in stat 3
         BaseAdd[3] = 0x300000;                  // base address for Caen_V1190B in stat 4
         BaseAdd[4] = 0x400000;                  // base address for Caen_V977   in stat 5



         Daq.setReg(BaseAdd[1] + 0x100A, 0x0);          //set interrupt level V1190B in stat 2
         Daq.setReg(BaseAdd[1] + 0x1022, 0x2);          //Almost Full Level   V1190B in stat 2
         Daq.setReg(BaseAdd[2] + 0x100A, 0x0);          //set interrupt level V1190B in stat 3
         Daq.setReg(BaseAdd[2] + 0x1022, 0x2);          //Almost Full Level   V1190B in stat 3
         Daq.setReg(BaseAdd[3] + 0x100A, 0x0);          //set interrupt level V1190B in stat 4
         Daq.setReg(BaseAdd[3] + 0x1022, 0x2);          //Almost Full Level   V1190B in stat 4

         Daq.setReg(BaseAdd[4] + 0xE,  0x0);    //interrupt mask for V977 (all channels enabled)
         Daq.setReg(BaseAdd[4] + 0x20, 0x7);            //interrupt for V977  enabled
         Daq.setReg(BaseAdd[4] + 0xA,  0xFF);           //set/reset the all output of V977
         Daq.setReg(BaseAdd[4] + 0xA,  0x0);            //set/reset the all output of V977

         Daq.RunLoop();
        }                                  // end main


//---------------------------------
        static public void DaqInit() throws Exception {
                // to be implemented
        }                          // end DaqInit  method

//---------------------------------
        static public void LamIsr() throws Exception {
```

```
                int Chan=0;
                int RawData=0;
                int[] EventStored = {0,0,0,0};
                int evStor=0;
                double Data=0;
                double LTim=0;
                double RTim=0;
                double LDist=0;
                double RDist=0;
                double X[]={0,0,0,0};
                double Y[]={0,0,0,0};
                double cosPhi=0;
                double cosTh=0;

                 double X0 , Y0;
                 double X1 , Y1;
                 double X2 , Y2;
                 double L00 , L10;

                 final double  W  = 50;
                 final double  L  = 100;

                 final double  Z0  = 100;            // z position of chamber
                 final double  Z1  = 0;
                 final double  Z2  = -100;

                 final double  Xd   = 42;           // sensible area
                 final double  Yd   = 85;
                 final double  Zd   = 0;

                 final double  strDist = 3.5;        // strip dist
                 final double  strWdt  = 2.5;        // strip width
                 final double  strLng  = 170;        // strip length
                 final double  tSpeed  = 4.5;        // strip signal speed 4.5 ns/m
                 final double  cSpeed  = 3.3;        // strip signal speed 3.3 ns/m
                 final double  strTim  = tSpeed * strLng / 100;

                nEv = nEv + 1;

System.out.println();
System.out.println("nEv  " + nEv);
System.out.println();

                EventStored[1] = Daq.getReg(BaseAdd[1] + 0x1020);    // retrieve the accumulate hit
number
                EventStored[2] = Daq.getReg(BaseAdd[2] + 0x1020);
                EventStored[3] = Daq.getReg(BaseAdd[3] + 0x1020);

                for (int j = 1; j <= 3; j++) {
                        evStor = EventStored[j];
```

```
                        for (int i = 1; i <= evStor; i++) {
                          RawData = Daq.getReg(BaseAdd[j] + 0x0);       // row data from TDC
                          Chan = ((RawData & 0x3F80000) >> 19);         // mask and get the channel
index
                          Data = (RawData & 0x7FFFF) / 10. ;            // get time value.

                // Divided by 10 because 0.1 ns/count
                          if (Chan <= 31) {
                                LTim = Data;                            // left channel
                          } else {
                                Chan = Chan - 32;
                                RTim = Data;                            // right channel
                          }
                        }
                // end for i

                        LDist = 0.5 * (LTim - RTim + strTim) / tSpeed;   // longitudinal position
(meter)
                        RDist = 0.5 * (RTim - LTim + strTim) / tSpeed;

//System.out.println("  data  " + Chan + "  " + LDist + "  " +   RDist  );
System.out.println("  data  " + Chan + "  " + LDist + "  " +   RDist + "  " + ( LDist + RDist));

                        X[j] = (Chan * strDist - Xd);                   // transversal position (cm)
                        Y[j] = LDist * 100 - Yd;
                }
            // end for j
System.out.println();


// --compute Cos(phi) and  Cos(theta)
                        L00 = Math.sqrt(Math.pow((X[1] - X[3]),2) + Math.pow((Y[1] - Y[3]),2));
                        L10 = Math.sqrt(Math.pow(L00,2) + 4 * Math.pow( Z0, 2));

                        if ((L00 == 0.) || (L10 == 0.)) {return;}

                        cosPhi = (X[3] - X[1]) / L00;
                        cosTh = 2 * Z0 / L10;

//                      H.hF1(cosTh);
                        H.hF1(cosPhi);

                        H.hShow();

System.out.format("CosTheta " +"%.2f%n", cosTh);
System.out.format("cosPhi   " +"%.2f%n", cosPhi);
System.out.println();

//                      Call EEE_frmHist.cntrHBook1.HF1(cosPhi)          // histogram
//                      Call EEE_frmHist.cntrHBook2.HF1(cosTh)
//                      Call EEE_frmHist.cntrHBook1.ShowHist
```

```
//                      Call EEE_frmHist.cntrHBook2.ShowHist

                        Daq.setReg(BaseAdd[4] + 0x10, 0x0);             //reset all channel interrupt
but Not the Mask reg
                        return;

        }                                       // end LamIsr  method


//---------------------------------
          static public void BrkIsr(String BrkMess) throws Exception {
        // to be implemented
                System.out.println(" BrkMess = " + BrkMess);
        }                       // end BrkIsr  method


//---------------------------------
        static public void GeneralMessage(String GenMess) throws Exception {
        // to be implemented
                System.out.println(" GenMess = " + GenMess);
        }     // end GeneralMessage  method


}       // end class EEE_Daq
```

7. External Signal Data File

The capability to accept input signals from files and send output signals to files has been introduced in TASS since Ver.3.0. This new feature allows the integration of trigger system in whole simulation chain of experiment.

Indeed, so far, the simulation of experiments has been limited to physical and geometrical aspects of detectors in order to maximize the efficiencies for the measurements under test. For this goal very powerful tools, like Geant program, are often used to trace the particles flow inside the experimental apparatus allowing to simulate hit and/or energy deposition in any elements of detector, but …

none information about resolution and dead time of trigger and data acquisition chain can be extracted from Geant output.

One of the goals of TASS is to fill this gap (see fig.1).

Indeed the Geant output file, correctly formatted, can be used as input signal data file to the TASS program simulating the real trigger hardware. TASS, in turn, produces an output data file, to be used by analysis program, reproducing the correct output from real data acquisition system.

No assumptions have been made about the resolution and data format, they are built-in TASS output. Just as example, the reader can try to imagine how a standard program can analyse two overlapping pulses coming from a poor time resolution detector; this problem is directly solved by TASS: it simulates the real hardware.

Another source of input data file could come from digital scope or digital signal analyser. Indeed the modern versions of these devices usually have the capability to store file of the acquired signals on a built-in mass memory device (USB memory, floppy disk etc.). These files, in turn, can be used as input data files in TASS environment.

Moreover, this new feature allows to partition very complex trigger system in separate blocks, each of them doing a specific job (for example, as in real world, block for energy trigger, muon trigger and so on). Each block produces its own output file then, merging those all together, they are sent as input file to final decision trigger block. The partition in several blocks allows to share the simulation among different teams and therefore to benefit a simpler and better design and tune of complex trigger system.

Input data files represent input signals that can be applied to any Lemo or Ecl input plug. User has responsibility to supply the correct data format and the correct amplitude (see later).

Output data files represent output signals and they can be issued from any Lemo and Ecl output plugs.

The input signal data files take the extension **.sif**
The output signal data files take the extension **.sof**

The output data files and input data files are format compatible, that is, one **.sof** file can be directly used as input **.sif** file

Let us understand, in following section, how TASS interacts with external data files.

The external signal data file format

TASS accept input signals data file **.sif** with format as reported in below table. The data file is organized in event block, each of them starting with the **[Block]** keyword and ending with **[EndBlock]** keyword.

You can think an event block as a collection of signals coming from the hardware detector and belonging to a single physical event.

Inside each event block, the **[Event]** lines define the effective input signals each one identified by its SignalName, SignalValue and SignalTime.

For every event block, many input signals can be arranged in the same event block, TASS takes care to reorganize in its memory all **[Event]** lines in ascending time order.

When TASS starts the run simulation, first of all, it checks if an input data file is requested (the **Tolls/Options/External Signals** tab defines the **.sif** file name) and, if this is the case, it collects the first event block.

When the first **[EndBlock]** keyword is detected, TASS starts the simulation using the **[Event]** data as input signal. When any signals belonging to the first block have been processed, TASS reads the next event block, processes it, and so on. If an EOF (end of file) is detected then TASS rewinds the file and runs again the same event blocks as new input signals. If **[Rewind]** keyword is missing no rewind operation will be performed and the run pauses at EOF; user is warned by the red led and the message "No more events" appearing on the tools bar of the main TASS window.

The **[Block]** keyword is optional, then it can be omitted, indeed it can be very useful to identify the BlockNumber if an error occurs.

Before starting the simulation the user has to associate any SignalName to its input plug, please refer to "**How to assign an input signal data file to input plug**" next section. In case a SignalName is not assigned to any plugs a warning message is issued.

---------- Signal Data File format ----------

```
[Comment],  Header comment line                                  (optional)
[Rewind]                                                         (optional)

[Block], BlockNumber                                             (optional)
[Event],  EvNum     , SignalName, SignalValue, SignalTime
[Event],  EvNum+1 , SignalName, SignalValue, SignalTime
[Event],  EvNum+2 , SignalName, SignalValue, SignalTime      } event block (BlockNumber)th
  .............
[Event],  EvNum+n , SignalName, SignalValue, SignalTime
[EndBlock]

[Block], BlockNumber+1                                           (optional)
[Event],  EvNum     , SignalName, SignalValue, SignalTime
[Event],  EvNum+1 , SignalName, SignalValue, SignalTime
[Event],  EvNum+2 , SignalName, SignalValue, SignalTime      } event block (BlockNumber +1)th
  .............
[Event],  EvNum+n , SignalName, SignalValue, SignalTime
[EndBlock]
```

The general structure of any lines inside the Signal Data File assume the format:

[Keyword], Param1, Param2, …

where Keyword is one of the following words and Param are the associate parameters depending from the keyword. [Keyword] and Parameters must be comma separated (CSV structure).

[Comment] keyword defines a comment line, the maximum length for the comment is approximately 2 billion (2^31) characters. Comment string has to be in a single line and separate by a comma from keyword. Many comment lines can appear anywhere in the file. TASS doesn't treat the comment lines but they can be useful to the user as marker point along the file.

[Rewind] keyword defines if the current **.sif** file has to be rewind and processed again when an EOF is detected.

[Block] keyword marks the start of an event block and defines the Block number. This is an optional line and the BlockNumber is reported by TASS if an error occurs reading the **.sif** file.

[Event] keyword defines the effective input signal.

EvNum identifies the n^{th} event line. The EvNum parameter is not treated by TASS but it can be useful to identify the event line if en error occurs.

SignalName is the name of input signal.

SignalValue is the value of input signal (value in Volt). The keywords NIM0, NIM1, TTL0, TTL1, ECL0, ECL1 can be used as well.

SignalTime is the current time (in ns), respect to the first event in the current Block, when a change occurs in the input value of signal

[EndBlock] keyword identifies the end of the current event block.

Note 1. Many keywords are optional but we suggest to include them because they can help you in debugging the file itself if an error in format occurs.

Note 2. The **.sif** file can be produced either by a program code (i.e. a suitable formatted output from Geant) or as result of TASS itself as a **.sof** file (see later).

Note 3. In TASS the analog signals are treated in a discrete approximation, meaning that signals level and time change only in step, rather than continuously. Therefore to describe an analogue signal many **[Event]** lines (each one being a change in the amplitude level of a signal) need to define as good as possible the signal itself.

Note 4. For each **EventBlock** TASS takes care to reorganize in its memory all **[Event]** lines in ascending **SignalTime** order.

How to assign an external input signal file to input plug

To use an external input signal file, before to start the simulation, the user has to associate the input signals to their input plugs.

First of all the user has to assign the Input Signal Data File **(.sif)** to the current project (the **.sif** path name is recorded by TASS when the project **.trg** file is saved):

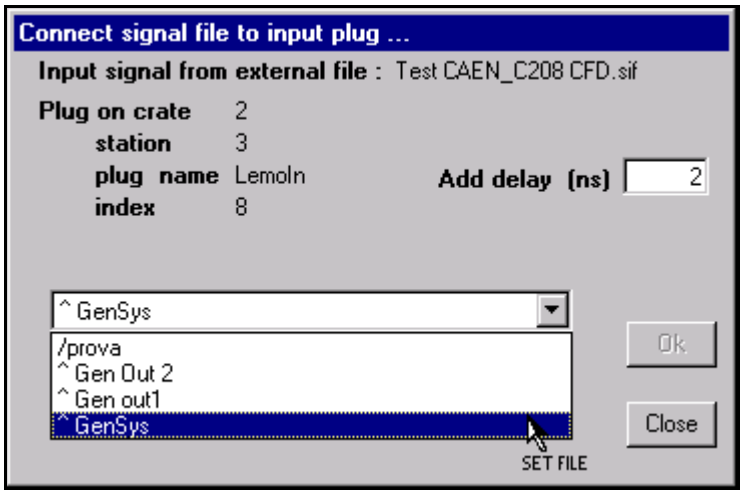
- Start TASS and load your trigger project file (.trg).
- Select **Tools/Options…/External Signals** tab, then select the **File (.sif)** option in the **Input signals from…** box.
  Note: leave the box empty if don't use External data input file

- Write the full path name in the text box or use the **Browse** button to navigate in the directory structure.
- Close the Options window by the **OK** button.

Once the signal file has been defined, TASS scans whole file and collects a list of all defined SignalName signals (see the previous section) contained in it.

Now you have to connect the input plugs to corresponding user defined SignalName.

Select the **External Signal/Connect to Input File** menu, the cursor change to SET FILE , click over the not connected input plug you wish to assign an input signal to. The form in following figure appears.



In the top part, the form shows the name of **.sif** file and the location of the input plug you have clicked.

Click on the combo text box, the list of all available input signals, defined into **.sif** file, is shown. Choose the signal you want associate to the selected plug. If you wish you can add a delay[*]**.**

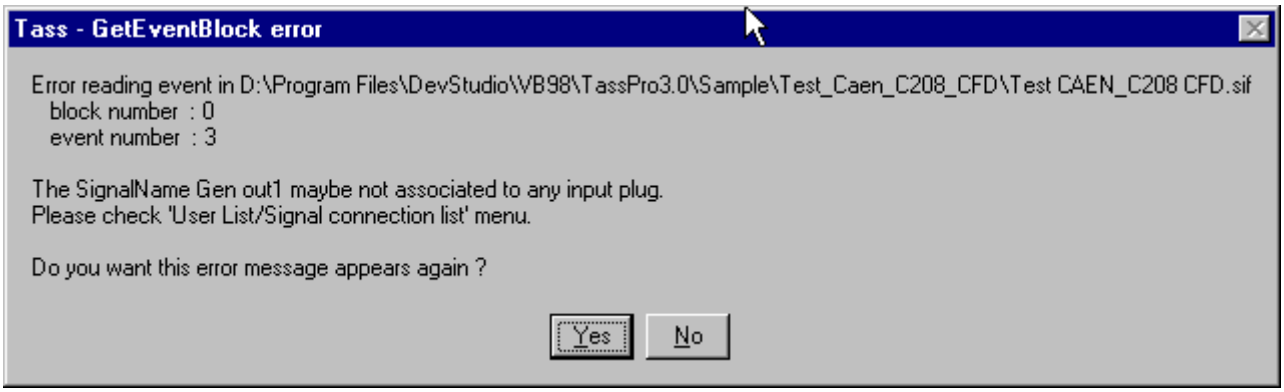
When a signal has been associated to a plug, it is marked "connected" by a leading character "^" and the plug itself changes its picture to *"connected to data file"* and it is marked by a yellow ring (similar behavior for Ecl plug, see **Cursor summary** section).

> **Note 1**: No more than one plug can be associated to every input signal, that is, only unmarked signals can be associated to free input plugs.
>
> **Note 2**: input signals can't be assigned to input plugs with a cable connection.

To remove an existing association choose **External Signal/Remove from Input File** menu then click on the plug you want disconnect. The **Remove from All Input File** drops down all the existing associations.

The list of all associations can be obtained by **UserList/Signal Connection List** menu.

You are not obliged to assign all the input signals to a input plug but, nevertheless, upon run starts TASS warns you by the message:



To avoid recursive warning messages you can answer "No" to the last question.

[*] The delay can represents the cable length connecting the plug, it will be add to SignalTime read from **.sif** file.

How to assign an external output signal file to output plug

To use an external output signal file, before to start the simulation, the user has to associate the output signals to their output plugs.

First of all the user has to assign the Output Signal Data File **(.sof)** to the current project (the **.sof** path name is recorded by TASS when the project **.trg** file is saved):

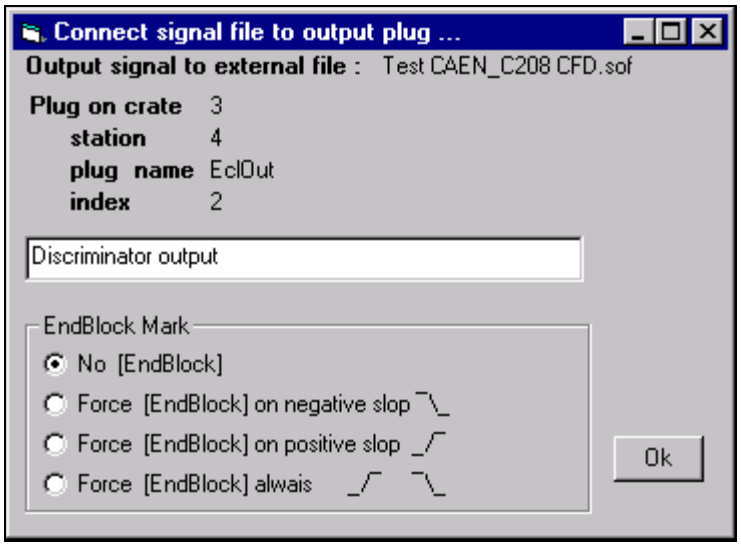
- Start TASS and load your trigger project file (.trg).
- select **Tools/Options…/External Signals** tab, then select the **File (.sof)** option in the **Output signals to…** box.
  Note: leave the box empty if don't use External data output file

- Write the full path name in the text box or use the **Browse** button to navigate in the directory structure.
- Close the Options window by the **OK** button.

Warning: if the **.sof** file already exists the new data will be appended to it.

Now you have to connect the output plugs to corresponding user defined SignalName.

Select the **External Signal/Connect to Output File** menu, the cursor change to SET FILE , click on the output plug you wish to assign an output signal to. The form in following figure appears.



In the top part, the form shows the name of **.sof** file and the location of the output plug you have clicked.

In the text box insert the name of signal you want to associate to the output plug.
You can choose if this signal, and its slope, has to be considered as the last event in the current event block. In this case an **[EndBlock]** keyword will be appended just after the current **[Event]** line on the **.sof** file (see "The external signal data file format" section).
The plug changes to *"connected to data file"* and it is marked by a yellow ring (similar behavior for Ecl plug, see **Cursor summary** section).

Note 1: The name of signal can't start with the leading special character "^".

Note 2: an output signal can be associated to an external file also if the plug is connected by cable.

To remove an existing association choose **External Signal/Remove from Output File** menu then click over the plug you want disconnect. The **Remove from All Output File** drops down all existing associations.

The list of all associations can be obtained by **UserList/Signal Connection List** menu.

Example program using external file data signal

In this example we show a simple case of an external data file use. User can consider the signal list in the .sif file as product by the Geant output or any other detectors simulation program.

User can try the external data file features running the trigger project set up **Test CAEN_C208 CFD_InputOutput_File.trg** provided in **Sample/Test/Test_Caen_C208_CFD** folder. In that you can recognize the same project **Test CAEN_C208 CFD.trg** where the NimWaveGen has been omitted. In this example, indeed, the input signals will be extracted from the external input signal file **Test CAEN_C208 CFD.sif** and sent to the input plug; the processed signals from the output plug are sent back to **Test CAEN_C208 CFD.sof** file.

The TASS simulation stops when the end of file is reached, to do a cycling run add, by hand, the **[Rewind]** keyword anywhere in the .sif file.
To know the assignments of any input and output plugs to the corresponding external signals use the **UserList/Signal Connections List** menu.

8. External Tcp/Ip signal data

Starting from Vers.4.0 has been introduced the new feature allowing TASS to be connect on the Internet network via Tcp/Ip protocol. This capability opens a new class of possible use of TASS system, that is:

TASS can accept input data representing external electrical signals and send output data via a Tcp/Ip internet connection(*). This new feature promotes TASS to become not only a simulator but also a very powerful monitor of hardware trigger system (see next "Remote Hardware Monitor" section).

As already mentioned at beginning of this section, TASS can accept data to the input plugs of the simulated modules and sends resulting simulation data on the output plugs via Tcp/Ip protocol. The input, output and control data will be simple string messages exchanged from server and client computer.
TASS will treat the input data as real electrical signals and it will process them to generate the resulting output data.

Although the data format for the Tcp/Ip communication is very similar to that of .sif file (see "**External Signal Data File**" section), some care has to be taken. Indeed in Tcp/Ip case the communication between the external world and TASS is organized as a server-client relationship and the communication itself has to be controlled by handshake messages.

The server (the computer suppling the data) has to provide one Tcp/Ip *listener* socket having the capability to send and receive ASCII string messages. The server uses the socket to exchange messages concerning values of input signals coming from detector and to provide the messages concerning data elaborate by the hardware trigger system.
User can follow the list of sample code in
Sample\TcpIp_Protocol\ TcpIp_DataFile_Server\TcpIp_DataFile_Server.vbp

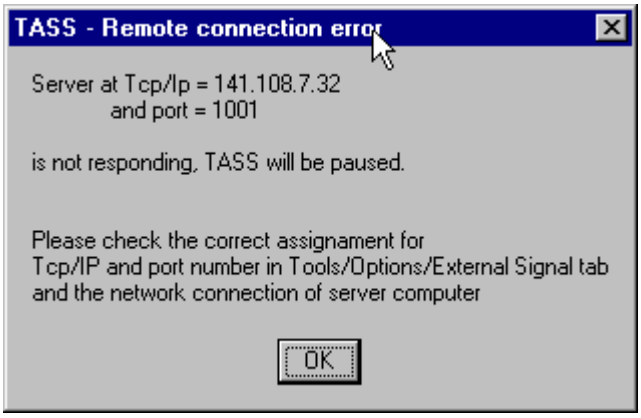
On the client side, TASS provides a built-in suitable socket. This socket provides logic to accept input data from Tcp/Ip connection and output as the real hardware does.

Of course, since the communication between the server and the client is based on the exchange of Tcp/Ip messages the two systems can run programs developed by different languages and to sit on different platforms, too. The only request is to provide the correct data format and the correct handshake.

(*) This function is similar at that described in the previous section "External File Data Signals". The difference is that in the those case the data came from an external file while here the data are provided through a internet tcp/ip communication.

In Tcp/Ip protocol, before any data transmission could take effect, a connection section has to be established in advance between the server and the client sides.

The client always starts the connection request. TASS takes six retry to attempt to establish the connection, after that, this error message is issued.



Indeed upon TASS starts the run simulation, first of all, it checks if an external Tcp/Ip connection is requested for the current project and, if the case, it tries to establish a communication with the computer which the Tcp/Ip and port numbers are defined into the **Input signals from…** box in **Tools/Options…/External Signals** tab. If the server grants the request the connection is established, otherwise an error message is issued and the run will be paused.

Note 1 : Of course a suitable program has to be provided on server side to handle the correct handshake messages and correct data format, refer to the example later in this section.

Note 2: The Tcp/Ip connection activity is displayed any time on the right side of task bar in the bottom part of main TASS window. Any time user can monitor messages exchanged by **External Signal** menu

- **Show messages on input socket**
- **Show messages on output socket**
- **Show messages on Daq socket**

Note 3: Since the connection is always started by client side, the server does not need to know in advance the client Tcp/Ip number: it takes this number from initial request for connection.

The external Tcp/Ip signal data format

TASS accepts input Tcp/Ip signals data with format as reported in following table.

The Tcp/Ip data consists of ASCII characters lines, each line starting with a keyword, enclosed in square brackets, followed by parameters, the numbers and type of them depending by the keyword. Keywords and parameters are comma separated (CSV)

The data lines are organized in event block, each of them starting with the **[Block]** keyword and ending with **[EndBlock]** keyword.

You can think an event block as a collection of signals coming from the hardware detector and belonging to a single physical event.

Inside each event block, the **[Event]** lines define the effective input signal each one identified by its **SignalName**, **SignalValue** and **SignalTime**.

For every event block, many **[Event]** lines can be arranged in the same event block, TASS take care to reorganize in its memory all the **[Event]** signals in ascending time order.

---------- Signal Tcp/Ip Data format ----------

```
[Comment],  Header comment line                                  (optional)
[Pause]                                                          (optional)

[Block], BlockNumber                                             (optional)
[Event],  EvNum    , SignalName, SignalValue, SignalTime
[Event],  EvNum+1, SignalName, SignalValue, SignalTime
  .....                                                          event block (BlockNumber)th
[Event],  EvNum+n, SignalName, SignalValue, SignalTime
[DataMessage], EvNum+n+1, Message
[EndBlock]

[Block], BlockNumber+1                                           (optional)
[Event],  EvNum    , SignalName, SignalValue, SignalTime
[Event],  EvNum+1, SignalName, SignalValue, SignalTime
  .....                                                          event block (BlockNumber +1)th
[Event],  EvNum+n, SignalName, SignalValue, SignalTime
[DataMessage], EvNum+n+1, Message
[EndBlock]
```

Note 1. Refer to previous section "The external signal data file format" for the mean of any parameters.

Note 2. The **[Pause]** keyword can be sent, in response to a **[GetEvent]** message, from the server to indicate that it is busy or data is not available (see below).

Note 3. The **[DataMessage]** keyword identifies any message. The DataMessage line is forwarded to the Daq program defined in the **Tools/Options/Daq** menu. It is user's responsibility to decode and use this messages string.

Note 4. The **BlockNumber** and **EvNum** parameters are not used in TASS, their purpose is only as data line marker useful to trace the line in case of error.

How to assign an external Tcp/Ip input signal to input plug

To use the external signals Tcp/Ip data, before to start the simulation the user has to associate the Tcp/Ip input signals to their input plugs.

First of all the user has to assign the Tcp/Ip server number to the current project (the Tcp/Ip number and all the associations are recorded by TASS when the project **.trg** file is saved):

- Start TASS and load or build your trigger project file (**.trg**).
- Select **Tools/Options…** menu, then the **External Signal** tab. Select in **Input signals from ...** frame the **Remote computer** option and write in the relevant text box the server Tcp/Ip and port numbers (default port is 1001).

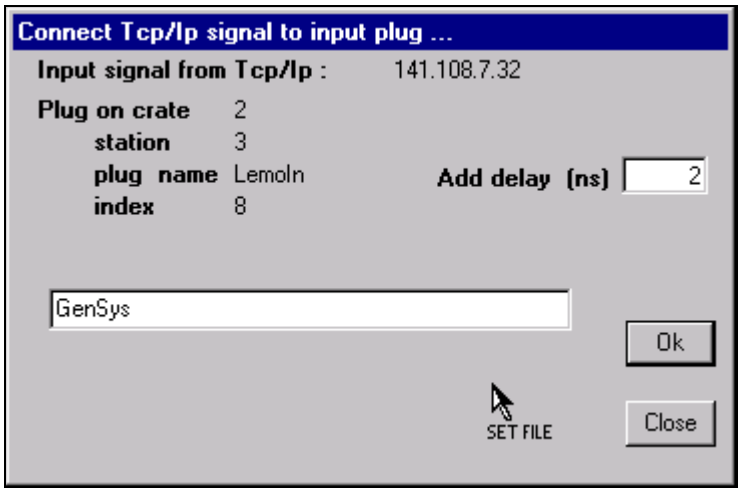
- Close the Options window by the **OK** button.

Now you have to associate the input plugs to corresponding data signal SignalName defined in [Event] line (see previous table).

On the main TASS window:

- Select the **External Signal/Connect to Tcp/Ip Input** menu, the cursor change to SET FILE

- Click on the input plug you wish assign an input signal, the form in following figure appears. In the upper part, the form shows the Tcp/Ip number of server computer and the location of the input plug you have click.



.

- Insert in the text box the **SignalName** chosen for the signal you want associate to the selected plug. If you wish you can add a delay(*).
- Repeat the previous procedure for all plug-signal associations.

Note 1: You can't assign a Tcp/Ip input signal to input plugs having cable connection.

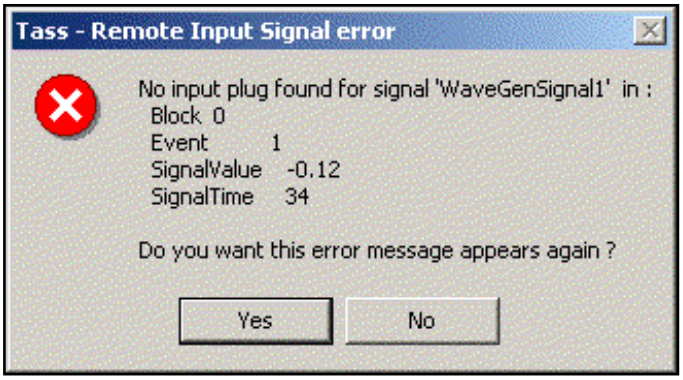
Note 2: In Tcp/Ip input signal case, TASS can't know in advance the name of all SignalName, then the user has the responsibility to assign to the input plug the correct SignalName that will be exchanged via Tcp/Ip messages.

To remove an existing association choose **External Signal/Remove from Tcp/Ip Input** menu then click on the plug you want disconnect. The **Remove from All Tcp/Ip Input** drops down any existing associations.

(*) The delay can represent the cable length connecting the plug, it will be add to SignalTime received from server.

TASS collects all the associations into its internal table, the list of all associations can be obtained by **UserList/Signal Connection List** menu.

You are not obliged to assign all the incoming *SignalName* to any plugs but, nevertheless, if TASS receives at run time an [Event] message with a *SignalName* not foreseen in its table then the following warning messages is issued:



To avoid recursive warning messages you can answer "No" to the last question.

Handshake protocol for Tcp/Ip input signal

Each **[Event]** line has to be retrieved from the server at the proper time (**[Event]** line by **[Event]** line basis) and for this purpose the following handshake scheme has to be followed (see "Handshake for Tcp/Ip input signal" flow chart in next page).

When TASS has no more events in its buffer to process, it sends a **[GetEvent]** message to the server to request the transmission of a new EventBlock, starts a timeout timer, then it enters in a waiting loop. This loop can be terminated only either by the reception the **[EndBlock]** or **[Pause]** messages from server or when the timeout timer expires. In this last case the simulation will be paused and a warning message issued.

Any time the server is collecting data into its Buffer and, if at least one Event is available, the **DataReady** flag will be set and a check on the **PendingReq** flag is done.

It is user responsibility to handle in the suitable way the server Buffer integrity, in case the server is written in Visual Basic then the *frmExternalData_Template.frm* can be used as guide.

Upon the server receives the **[GetEvent]** message it set the PendingReq flag.

PendingReq = True means a preavious data request has not been till granted then the server verifies its buffer to see if a physical data event block is ready to be transmitted:

a) Data is not ready. This can happen if the server is busy doing something else, for instance, the physical run has been paused or the physical event is not still collected, and so on.
In this case the server sends back a **[Pause]** message to warn the TASS client to wait.

b) Data is ready. This means the physical data event block has been collected in the server buffer and it is ready to be sent.
In this case the server sends the first data line (see previous tab).

When TASS receives the message it checks its validity and analyses its meaning:

a) **[Pause]**. This message means data is not ready to be transmitted by server then TASS stops the timeout timer and waits.

b) **[Event]**. This message is an event data signal, TASS collects it in its data buffer then sends back a new **[GetEvent]** message to request the next event data line.
The **[GetEvent]- [Event]** handshake process goes on until all events data signal in server buffer have been transmitted. The **[EndBlock]** message (see later) stops this handshake.

c) **[DataMessage]**. This special message is issued from the server to send any information to the Daq program. Indeed, TASS doesn't treat this message but simply forwards it to companion Daq program, is user responsibility to decode and take proper action in response to it. Any number of **[DataMessage]** lines and anywhere in the block can be sent.
Usually the server uses the **[DataMessage]** messages to pass value of real hardware Camac/Vme register or results of simulation procedures or to warn for special conditions run.

d) **[EndBlock]**. This message means all events data signal of current Block have been transmitted then TASS can start the simulation.

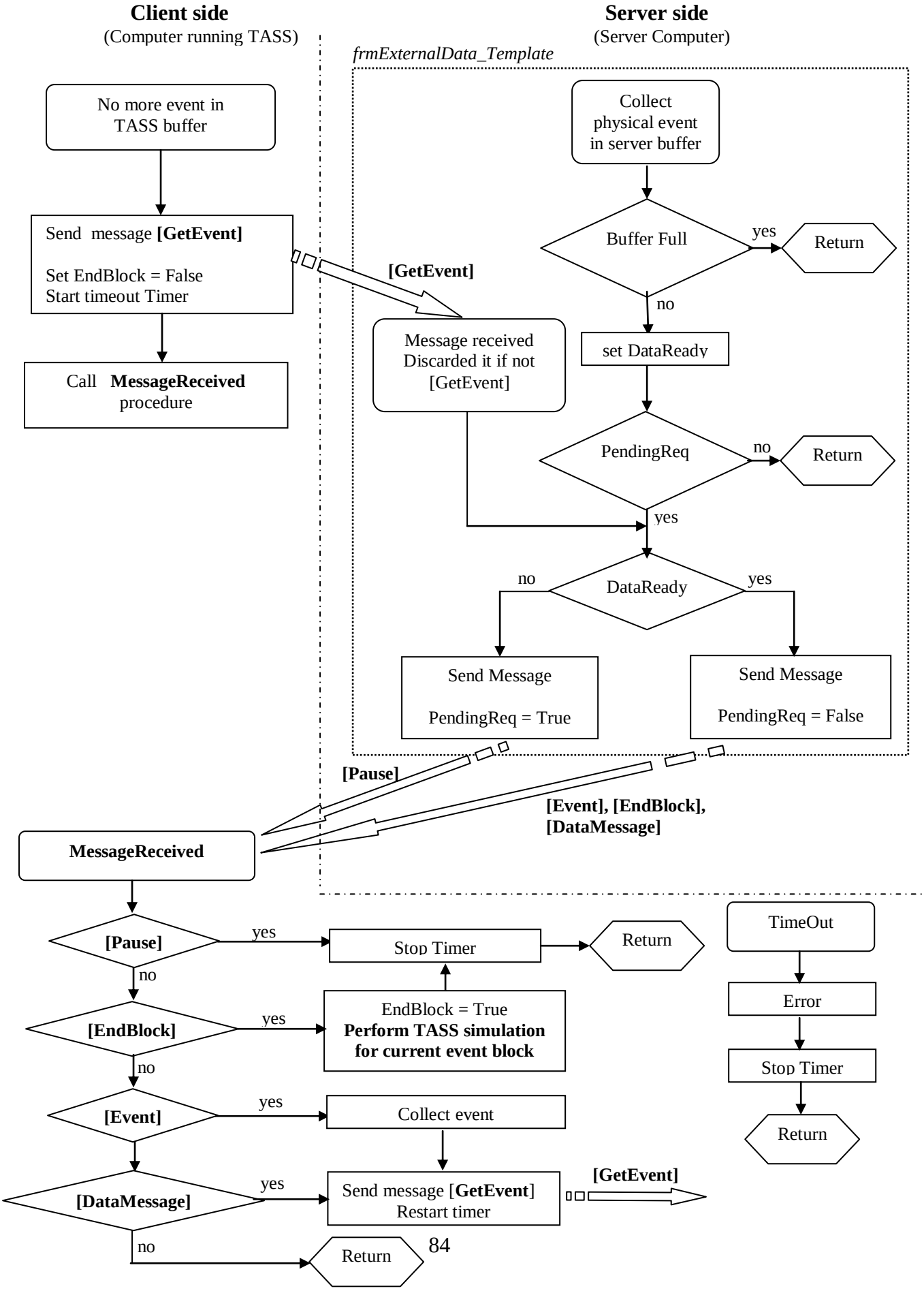
e) Any other messages will be discarded.

Upon **[EndBlock]** has been received, TASS starts the simulation using the **[Event]** data collected in its buffer as input signals. When any signals belonging to the current event block has been processed, TASS sends a new **[GetEvent]** message to request the next block, processes it, and so on.

Note: For debug and monitor purpose user can select **Write received messages** options in the **External Signal** menu. This feature writes on the logger window all the received messages allowing the user to trace and debug the exchanged data and handshake messages between server and client.

The data flow in following picture represents in graphical way the previous explanation.

Handshake for Tcp/Ip input signal



How to assign an external Tcp/Ip output signal to output plug

In some case can be needed to send signals or result data produced by TASS simulation to a third computer. TASS has built-in all the support for this purpose.

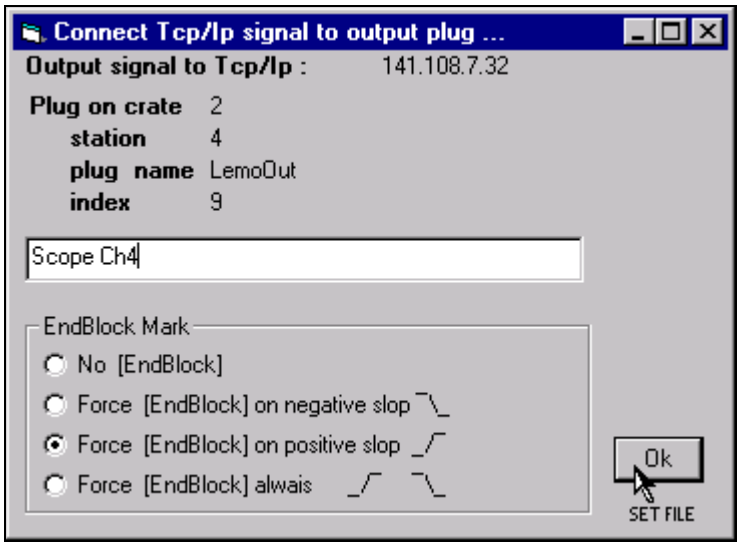
To use an external Tcp/Ip output signal, before to start the simulation, the user has to associate the output signals to their output plugs.

First of all the user has to assign the output signal Tcp/Ip server number to the current project (the Tcp/Ip number is recorded by TASS when the project **.trg** file is saved):

- Start TASS and load your trigger project file (.trg).
- Select **Tools/Options…** menu, then the **External Signals** tab. Select in **Output signals to ...** frame the **Remote computer** option and write on the relevant text box the server Tcp/Ip and port numbers (default port is 1002).
- Close the Options window by the **OK** button.

Now you have to connect the output plugs to corresponding user defined SignalName.

Select the **External Signal/Connect to Tcp/Ip Output** menu, the cursor change to SET FILE , click on the output plug you whish assign an output signal, the following figure appears



In the upper part, the form shows the Tcp/Ip number of server computer and the location of the output plug you have clicked.

Insert in the text box any name of signal you want associate to the selected plug.

You can choose if this signal, and its slope, has to be considered as the last event in the current event block. In this case an **[EndBlock]** message will be sent, just after the current **[Event]** line to the server computer (see "The external signal Tcp/Ip data format" section).

Note 1: The name of signal can't start with the leading special character "^".

Note 2: an output Tcp/Ip signal can be associated to a output plug also if it is connected by cable.

To remove an existing association choose **External Signal/Remove from Tcp/Ip Output** menu then click over the plug you want disconnect.
The **Remove from All Tcp/Ip Output** drops down any existing associations.
The list of all associations can be obtained by **UserList/Signal Connection List** menu.

In Tcp/Ip communication each **[Event]** line has to be sent to the server at the proper time (**[Event]** line by **[Event]** line basis) and for this purpose the following handshake skim has to be followed.

When TASS has an output data to be sent, it issues an **[Event]** message to the server and starts a timeout timer then enters in a waiting loop. This loop can be terminated only either by the reception the **[Ack]** message from server or when the timeout timer expires. In this last case the simulation will be paused and a warning message issued.

From the server side point of view, the reception of the **[Event]** message starts the following procedure. The server checks the PendingAck flag to see if a previous message is still not granted, then it verifies if itself can accept more incoming messages:

a) **Server is busy**. This can happen if the server is doing some thing else, for instance, the physical run has been paused or the event buffer is full, and so on.
In this case the server sends back a **[Pause]** message to warn the client TASS to wait.
b) **Server is ready**. This means the event message can be collected in the server buffer.
In this case the server sends an **[Ack]** message to notify to client TASS the message has been accepted and processed.

Note: After a **[Pause]** message the server, when ready again, has to send an **[Ack]** message in order to recover the connection.

Upon TASS receives the back message it checks its validity and analyses its mean:

a) **[Pause]**. This message means server is not ready then TASS stops the timer and waits.
b) **[Ack]**. This message means server accepted the previous **[Event]** message and it is ready for the next one. TASS stops the timer and prepares for next output data.
c) Any other messages will be considered optional and then discarded.

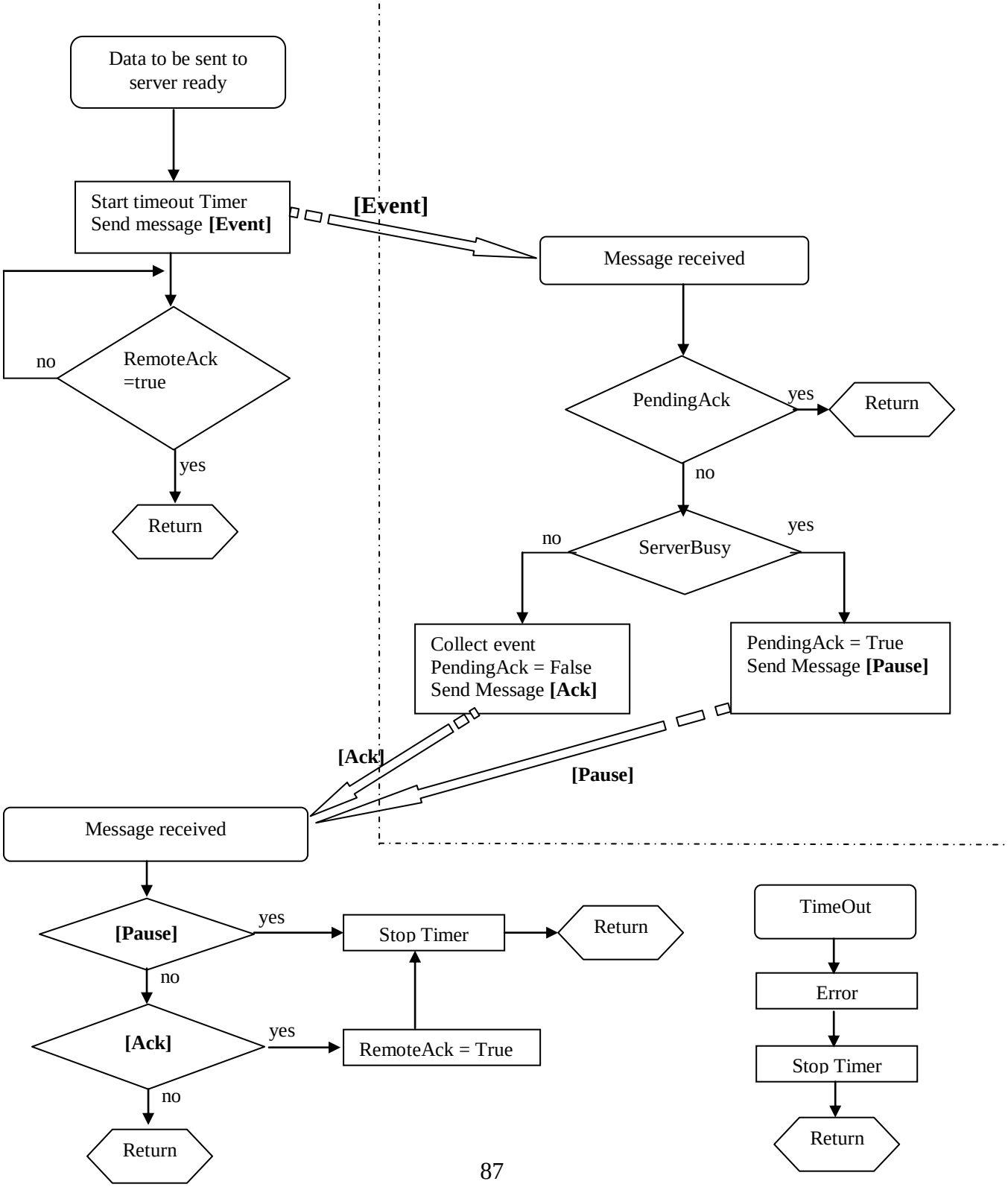
Note: For debug purpose user can select **Write sent messages** options in the **External Signal** menu. This feature writes on the logger window all the sent messages allowing the user to trace and debug the exchanged data and handshake messages between server and client.

The data flow in following picture represents in graphical way the previous explanation.

Handshake for Tcp/Ip output signal

Client side
(Computer running TASS)

Server side
(Server Computer)



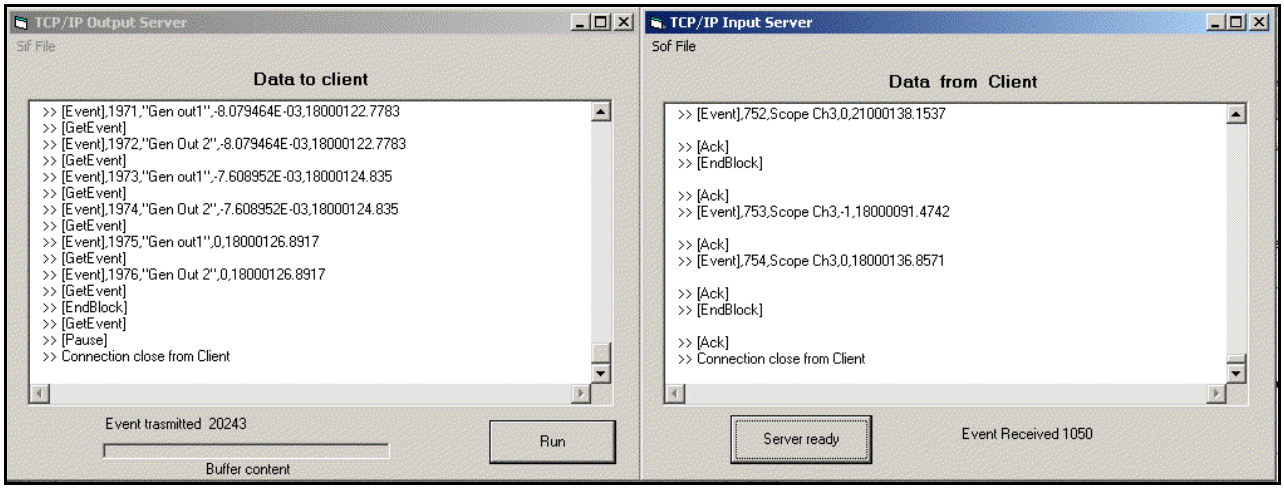
Example programs for external Tcp/Ip data signal

Here we report explanation and comments about a simple program showing how to use the Tcp/Ip features of TASS system.

The user, after installation of TASS package, can find the code programs for these examples in /Sample/TcpIp_Protocol folder.

Tcp/Ip DataFile Server

This is a very simple example using the external Tcp/Ip data signal features provided by TASS system. It shows only how to implement the communication between server and client without any user software implementation, no any interaction with real hardware is taken in account and, indeed, the external signals will be extracted from a data file.



In this example the real external data will be simulated by data extracted from the file **Tcp_Ip_DataFile.sif**(*) (**), and sent via network to client where TASS will process them to produce the output data that will be sent back to the server and stored in **Tcp_Ip_DataFile.sof** file.

Both files (.sif and .sof) are user selectable by menu.

When the whole set of data in the .sif file has been used then the file will be rewind and the same set used again as new data.

On server side the running program simulates a very fast dummy data acquisition and, when the client TASS asks for a new event, the next current event will be sent via Tcp/Ip connection.

Two buttons, one for each connection, allow the user to pause the transmission to simulate a more realistic behaviour of data acquisition system.

(*)Refer to **External signal data file** and **How to assign an external file output signal to output plug** sections to understand this feature.

(**) The data file **Tcp_Ip_DataFile.sif** can represent, for instance, data produced by Geant output simulating our detector.

To try the program go in the following step:

On the server computer.

1. Into **Sample\TcpIp_Protocol\TcpIp_DataFile_Server** folder start the **TcpIp_DataFile_Server.exe**(*) program.

2. By the **Sif File** menu in the left window navigate in the system directory and select the **Tcp_Ip_DataFile.sif** file.

3. By the **Sof File** menu in the right window choose or write the file name where the data sent back to the server will be stored.

4. Push the **Run** button. The **Buffer content** bar shows the current level of the server buffer. The **Event transmitted** label shows the number of simulated events sent to the client computer running TASS program.

On client computer.

1. Start **TassProject.exe** program.

2. Select **File/Open…** menu, then navigate in the system directory and select the trigger file **/Sample/TcpIp_Protocol/TcpIp_DataFile_Server/ ExternalDataSignal.trg** (to save packaging space this file is included into "server" folder but it represents the client side)**.**

3. Select **Tool/Options…** and in the **External Signals** tab make sure both the **Remote computer** option boxes are selected, then write down the correct Tcp/Ip address and port number of your server computer, then close the window.
The Tcp/Ip number of server computer is shown in blue character in the right form. If user uses the same computer for client and server then has to insert the Tcp/Ip number **127.0.0.1** or "**LocalHost**" name.
The default port numbers are **1001** and **1002** for the *External Input Signals* and *External Output Signals* respectively.

4. Start the simulation by the **Run/Cont** button. TASS asks for a Tcp/Ip connection to the server and you can see the connection status on the tooltip on the right side of the StatusBar of main TASS window.

(*) If the Visual Basic 6 system is present on the server, you can edit and modify any part of code (refer to Microsoft Visual Basic documentation), open TcpIp_DataFile_Server.vbp.
User could translate the present program in any language he likes and running it under any operating system. In this case, of course, the Visual Basic presence is not mandatory.

If everything is ok, you should see in the left window of server the messages sent to TASS and on the right window the messages received.

For every received **[EndBlock]**, TASS process the events in the current block and you can see the resulting behaviour on the digital scope.

You can simulate a more realistic data acquisition behaviour by the buttons **Run/Pause** and **ServerReady/ServerBusy** in both windows.

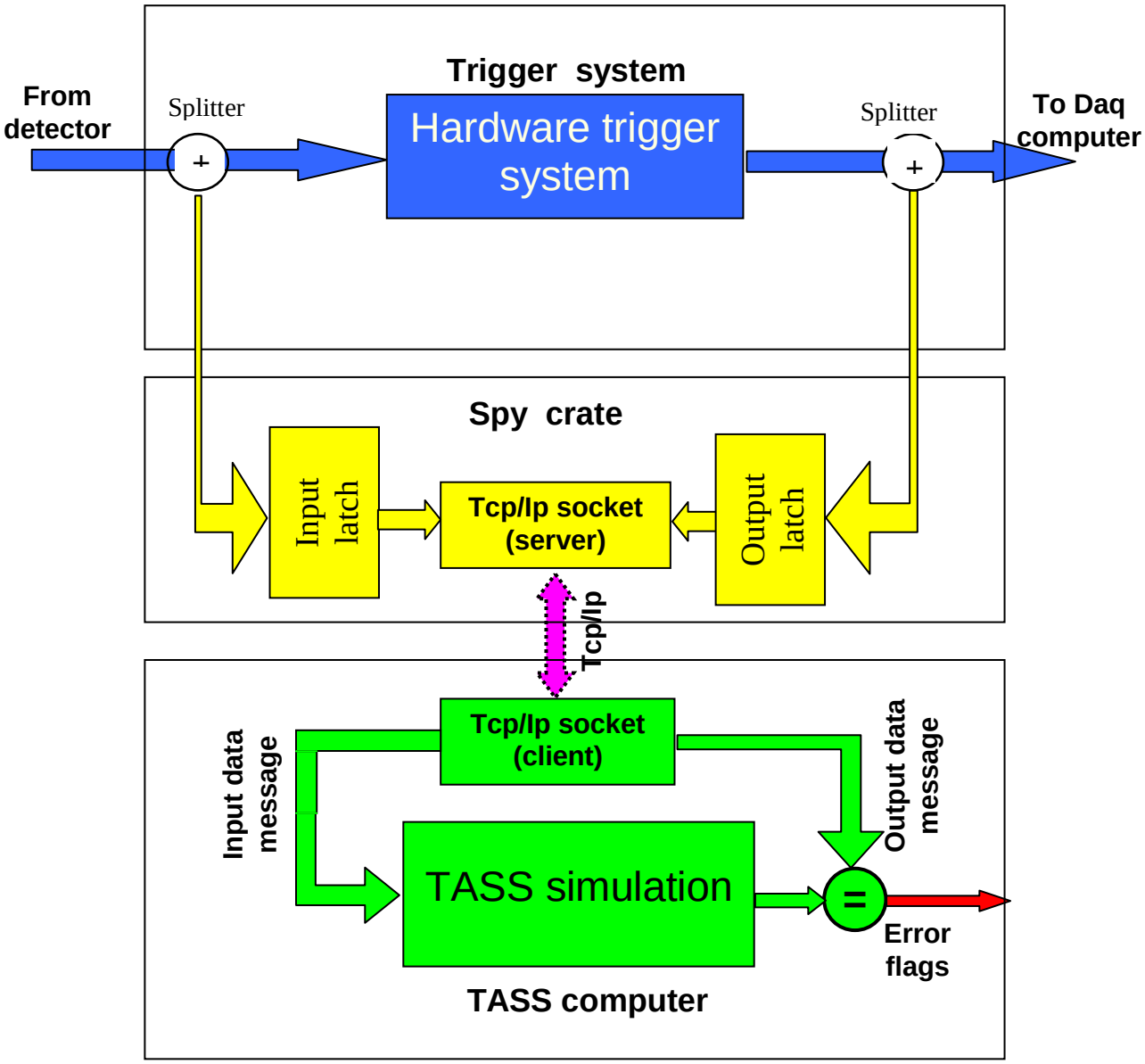
Take into account that the server will stop to sending data only when the buffer is empty.

Note: In the Tcp/Ip protocol before to exchange any data a connection has to be established between server and client. It is always the client requiring a connection to the server, this means you need to have the server already running and ready to accept the request.

9. Remote Hardware Monitor

Note: in the present version of TASS manual no any example will be shown for the Remote Hardware Monitor in this section. Let's postpone to next version to present this feature.

TASS can accept input data representing external electrical signals and send output data via a Tcp/Ip network connection. This new feature promotes TASS to become not only a simulator but also a very powerful monitor of real hardware trigger system.



Usually the event signals coming from the detector are issued to the hardware trigger to be analysed and, on the basis of physical request, rejected or accepted to be sent to the Daq system computer (dark/blue path in top, without splitter). An hardware fault on this chain can be discovered

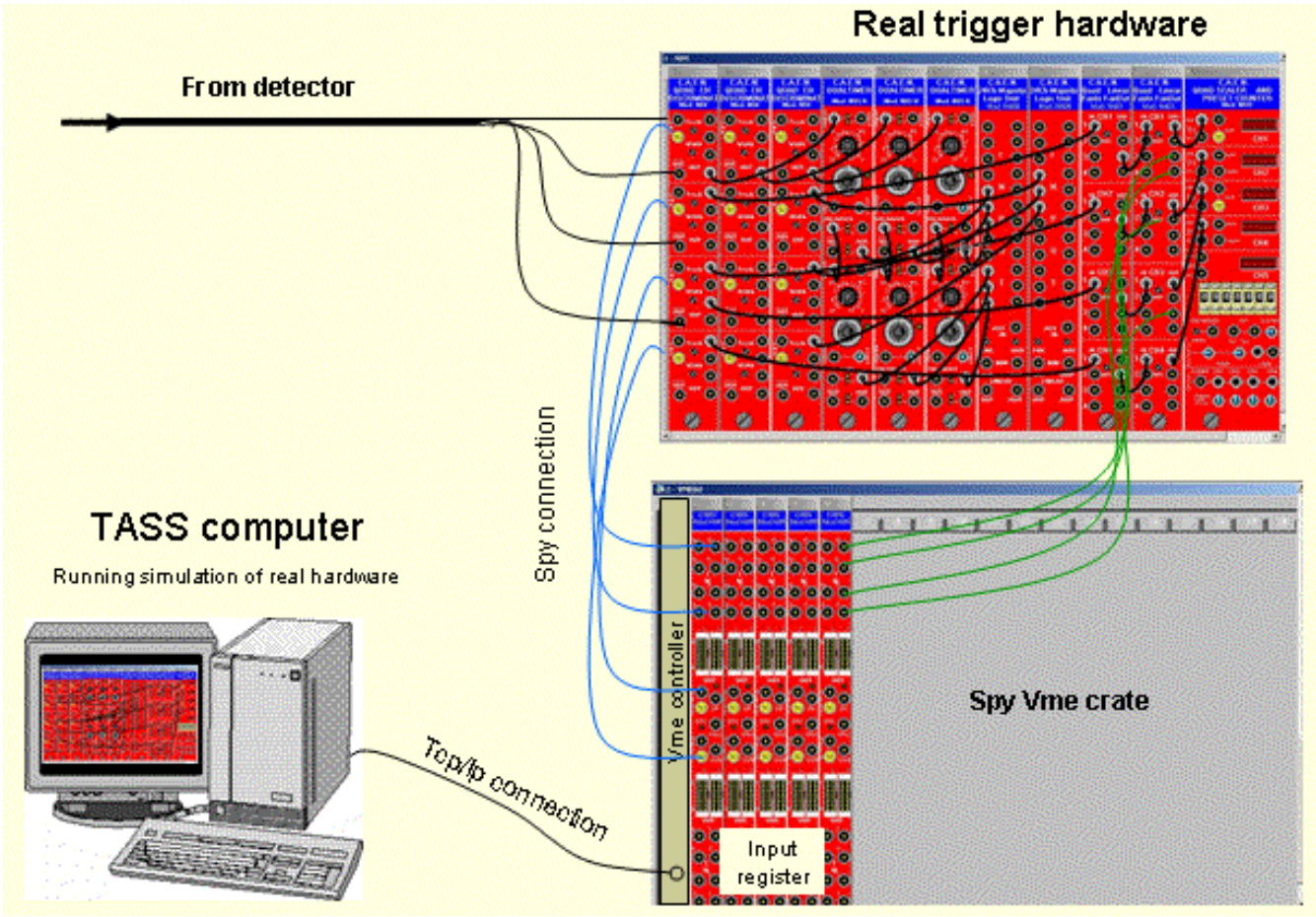
only on the basis of the analysis of physical data collected, often very long time after the fault occurred, this means unreliable data or complete data loss.

Therefore a prompt discovery of bad hardware functionality is a mandatory request.

TASS can solve this problem as shown in figure (light/yellow path, with splitter): suppose TASS simulates the real hardware trigger. A sample of input signals from detector (on physical event basis) can be latched and converted as data and sent, via a Tcp/Ip communication, to TASS program (green path) where they will be treated as in the real hardware. The TASS output can be compared with the latched output from the trigger system then flags can be set. A difference between real output and the simulated one can therefore warn the operator!

Of course, due to the different speed of the paths, the comparison can be done only on a sample of events, but the delay involved to alert the user for a hardware fault can be of order of few tens of seconds for complex trigger system simulation.

The above principle scheme can be implemented, for instance, in the following way



In figure the above crate represents the real trigger system and the electrical signals coming from detector are split in two branches, the first one going to the relevant input of trigger and the second (blue cables) going to the input registers (latch) system located in a spy crate. The output signals of trigger are collected (green cables) in another input registers (latch) system located in the spy crate, too.

The Vme controller takes cure to format the readout input and output signal values in suitable way to be sent on the network (see "External Tcp/Ip Signal Data" section). On the client side there is a TASS simulation reproducing the trigger setup and doing the proper comparison.

To be note that none changes are done to the real system (Blue path)! This is a mandatory request.

Being the communication based on Tcp/Ip exchange of messages, the monitor process can be done on world wide remote sites.

Even more: to know a fault occurred is not enough; you have to find where the fault is!

TASS can be extremely useful to help the trigger designer to identify a bad working module in the hardware system doing the cross check between the real set up and the simulated one. Indeed the tools supplied by TASS (scope, wave generator, running step by step, and so on) allow to perform this cross check in easy and efficient way.