



INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE: CENTRATOM TRIESTE



**The United Nations
University**

SMR/774 - 13

**THIRD COLLEGE ON MICROPROCESSOR-BASED REAL-TIME
CONTROL - PRINCIPLES AND APPLICATIONS IN PHYSICS
26 September - 21 October 1994**

LABVIEW

**Fabio SOSO
C.E.R.N.-European Organization for Nuclear
Research
C.N. Division
1211 Geneva
SWITZERLAND**

These are preliminary lecture notes, intended only for distribution to participants.

Software Advances in Measure- ment and Instrumentation. LabVIEW

Abstract:

Today's data acquisition and instrument control systems combine personal computers (PCs) and workstations with sophisticate software that integrates general-purpose data-acquisition, data analysis, and intuitive, graphical data presentation.

Fabio Soso, CERN-CN/CE
soso@cernvm.cern.ch
October 1994

INDEX

- 1- Subject of the talk
- 2 - Playing with a Bouncing Cube
- 3 - Electronic instrumentation today
- 4 - Structure of programs for instrumentation control
- 5 - The ideal software
- 6 - Development of LabVIEW
- 7 - Characteristics of LabVIEW
- 8 - Conclusion
- 9 - Annex
- 10 - Slides

Acknowledgement: I'm indebted to Gary Johnson for his new, interesting, book on LabVIEW. Information therein was very useful, particularly about early years, and worded out better than I could have done.

1 - SUBJECT OF THE TALK

I wish to present LabVIEW¹, a program developed by National Instruments Corporation of Austin, Texas, for the control of electronic instrumentation.

Fortunately, LabVIEW is more than a specialized application: it's a purely graphic, general purpose programming language, with extensive libraries of functions and an integral compiler and debugger. It has few intrinsic limitations, and only a minimal performance penalty when compared with conventional programming languages. The fact that LabVIEW is a real programming language, of anew kind, makes it an interesting topic.

To evaluate LabVIEW in its natural context of instrumentation control, we must consider two introductory subjects:

- (a) electronic instrumentation: it evolves rapidly, presenting the user with an apparently chaotic set of choices.
- (b) software for instrumentation control; this generic term includes quite a variety of tasks and solutions.

In fact, the situation is so fluid, that we'll see that instruments are not instruments any more - we'll call them "virtual instruments" - and the software too can be different from the traditional programming languages.

The introductory parts, although useful for the general understanding, risk to be slightly boring. Hence to keep interest alive, we will start with an amusing (yet instructive) LabVIEW demonstration.

Let us play with a Bouncing Cube (Fig 2.1).

1. Laboratory Virtual Environment Engineering Workbench

2 - THE BOUNCING CUBE EXAMPLE

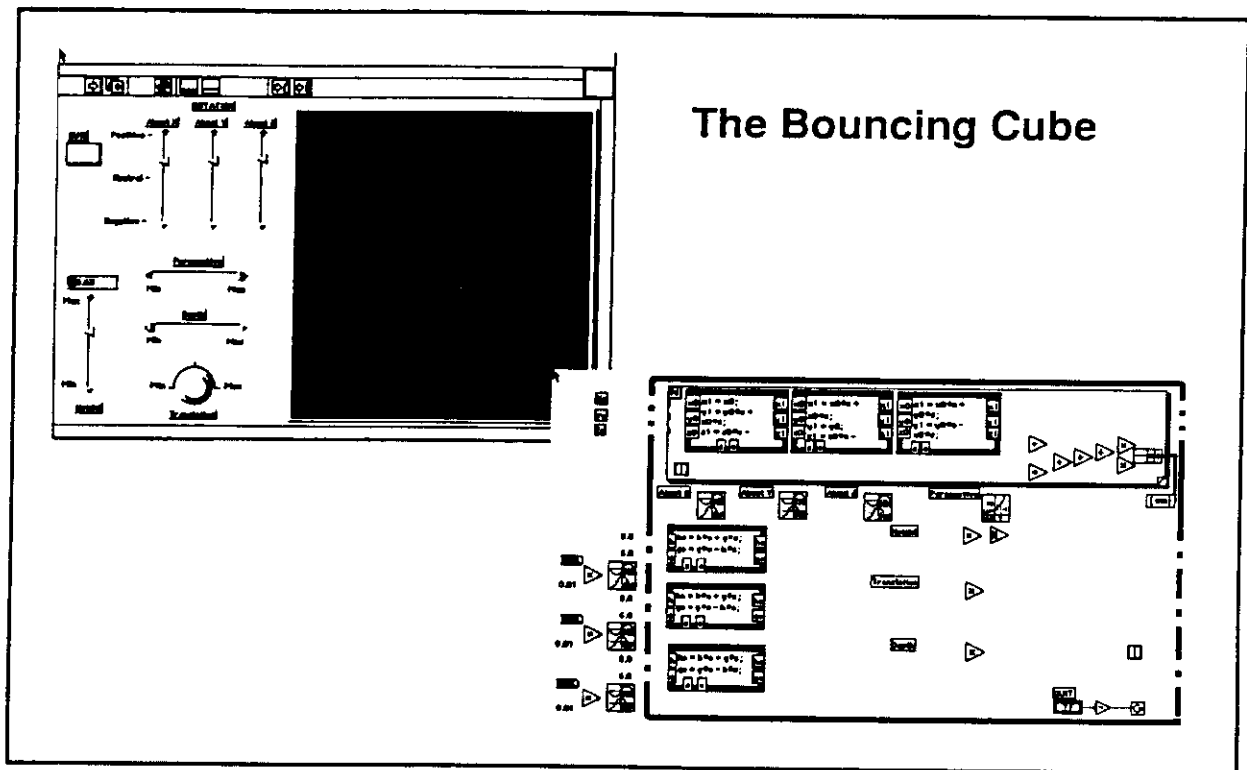


Figure 2.1 A LabVIEW Example

By playing with the bouncing cube (as with any other Virtual Instrument) we can appreciate some LabVIEW features:

- a user friendly, easy to customize "front panel" interface;
- an easy to read, self documenting logic diagram;
- the Help feature;
- the built-in, light-speed graphic compiler, immediate check for syntax errors;
- the monitoring/debugging features when running an application;
- the multitask capability of LabVIEW;
- The portability of applications. For instance, the binary file describing the Bouncing Cube for Windows could be e-mailed to a MAC or SUN user, and run without modifications.

3 - ELECTRONIC INSTRUMENTATION

In the 70's, instruments were mostly "analog". Power supplies, multimeters, sinewave/pulse generators, fast or slow oscilloscopes, were controlled by knobs, switches, displays on the front panel; the user couldn't modify the type and number of controls, nor the instrument built-in functions.

The recording of measurement was essentially manual (pencil&paper, Polaroid cameras, occasionally a chart recorder).

In the 80's, instruments became more sophisticated with the addition of numerical functions. Semi-automatic measurement recording used printers, paper tape punches, mag tape drives; some instruments had a special connection to a small computer equipped with a teletype terminal. Data recording was followed by off-line data processing. The functional characteristics of instruments were still those defined by the manufacturer, every bit of extra flexibility adding to the cost.

Two events marked this time:

- (1) The definition, by Hewlett Packard, of a General Purpose Instrumentation Bus (GPIB) for the connection and control of groups of instruments; GPIB soon became an international standard, ref. IEEE 488, and is still in full use.
- (2) The PC revolution. As the processing power of desktop computers surpassed the internal processing capability of most instruments, the benefits of connecting PC's to instruments increased as well, and so did the number of users.

Control and data acquisition tasks shifted from the controlled instrument to the controlling computer.

The instruments began to change, some front panels lost their buttons, replaced by graphic windows with operating menus on the computer screen.

For simple applications, the whole instrument could be replaced by a single plug-in board, inside the computer.

Today, we can use various types of instruments, more or less complex, more or less costly, but never disconnected from a computer, which has become the core of a measuring system (Fig. 3.1).

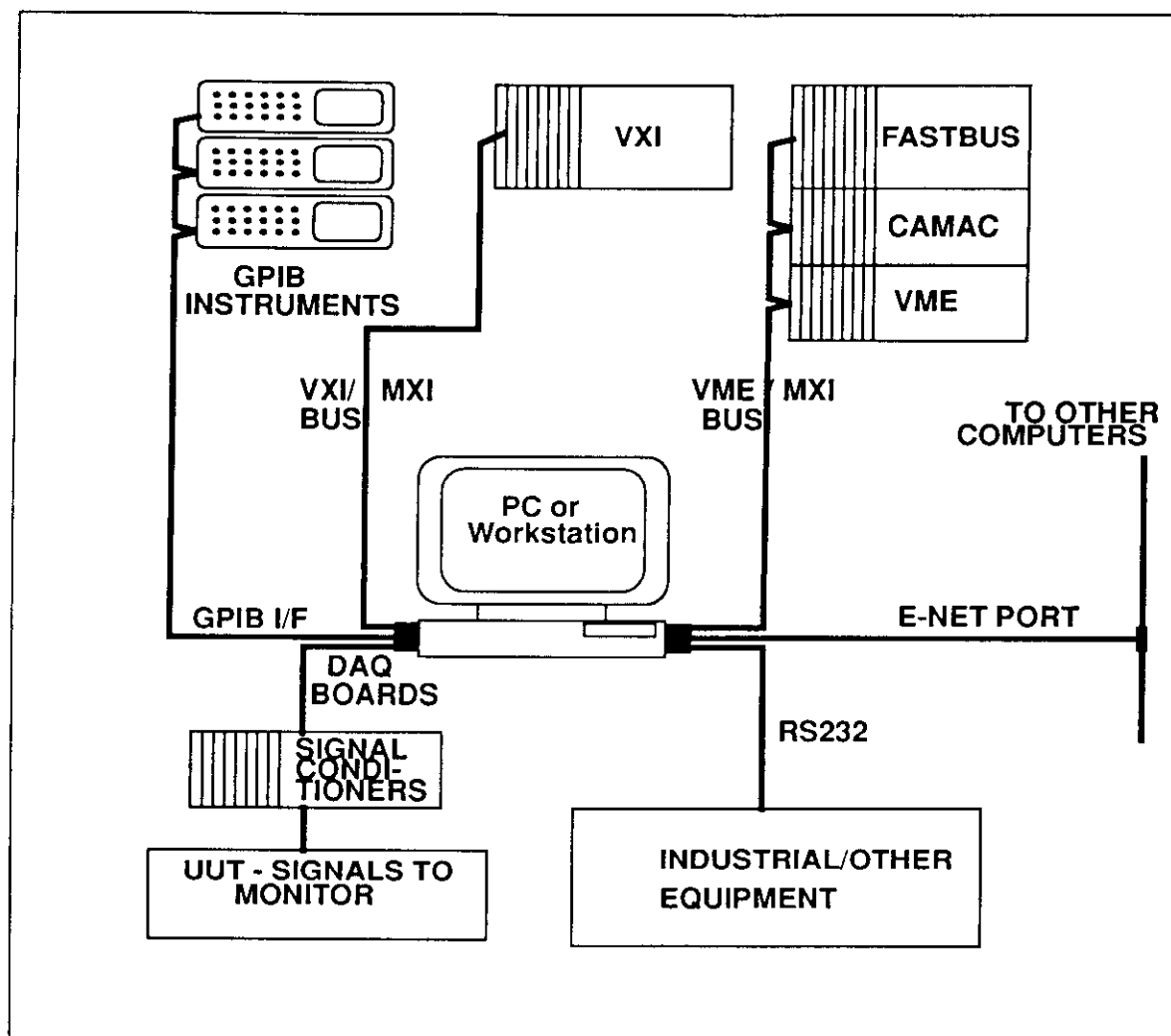


Figure 3.1 Instrumentation Control System

Schematically, we may find:

- (1) A PC or a workstation, with some interactive software for instrumentation control.
- (2) Traditional style instruments, often performing and costly, with crowded front panels, internal memory and signal processing capabilities (digital oscilloscopes, digital multi-meters, frequency synthesizers, logic analyzers, etc.). The instruments (Slide 2a,2b) are connected to a computer via the GPIB bus, or via a serial port. The front panel can be replicated on the computer screen.
- (3) Modular instruments (Slides 5b to 7a), i.e. plug in modules of a standard chassis; types include VME or VXI, Camac and Fastbus in physics experiments. The front panel is reduced to the bare minimum, every control and display

being provided by the computer. Modular instruments are smaller, easier to test and develop, user-configured: the choice of functions, the number of channels, the amount of memory, the use of local intelligence can be tailored to the user's needs and modified at any time.

Moreover, the various instruments manufacturers are striving to define universal routines for the control of instruments (SCPI standard commands for instrumentation programs, Plug&Play for VXI modules and plug-in boards)

- (4) Plug-in data acquisition boards for the various types of computers (Slides 3a to 5a). The evolution of Analog to Digital converter circuits and of signal conditioning modules makes the choice of plug-in DAQ boards a popular one (whereas VXI and GPIB instruments are used for more sophisticated measurements).

Signal conditioning modules are used to adapt the transducer electrical signals to the DAQ boards input: amplify, linearize, isolate, filter. Also multiplexing is a useful technique when measuring many signals.

- (5) Programmable Logic Controllers (PLC's), robust and reliable low speed analogue and digital I/O interfaces and controllers for industrial equipment. Used in physical plants, they use proprietary series protocols, many of which are supported by LabVIEW drivers available from 3rd-party developers.

The future will bring us new features. Fig 3.2, for instance, shows a distributed measurement and control system developed by an Australian manufacturer of implanted cardiac defibrillators.

The control system is used to set up instruments, initiate tasks, collect data in a useful format for the final report of each produced part.

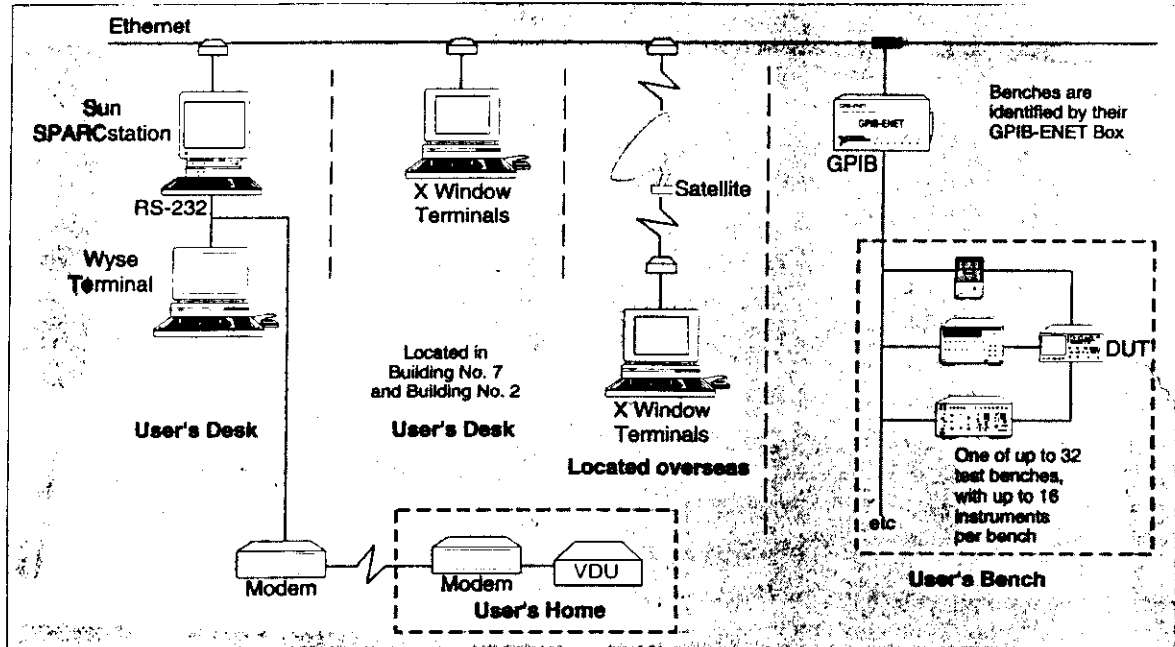
By using Sun Sparcstations, GPIB/ENET interfaces the size of a cigarette box, and Tektronics TDS series oscilloscopes, the full system can control up to 32 test benches, each containing up to 16 test instruments.

In this multitasking environment, the company engineers, without leaving their desk, can use their workstations to connect to any group of instruments via the GPIB/ENET interface and to check the performance of a device under test.

Operators can also control and monitor tests and review data at home, via a modem, or anywhere in the world with Ethernet.

Another trend, towards miniature, portable control systems, is shown by PCMCIA data acquisition and interface boards for

notebook and Laptop computers (Slide 8a). These boards have the same performances of their full scale correspondents, but with an extremely low power consumption and the size of a credit card.



By integrating the GPIB-ENET interface kit into the ICD test and measurement system, Teletronics operators can test ICDs without leaving their desks, or review data from anywhere in the world via Ethernet.

Figure 3.2 A distributed Instrument Control System

4 - STRUCTURE OF PROGRAMS FOR INSTRUMENTATION CONTROL

Fig. 4.1 gives a schematic view of a computer controlled test&measuring system, including the various parts of the control software.

(From bottom to top). Physical parameters are transformed into electrical signals by transducers/actuators, with the help of signals conditioners/multiplexers when required.

The electrical signals are measured, i.e. converted to numerical data, by various types of instruments: stand alone, modular, DAQ's (data acquisition boards), industrial controls.

Each type of instrument is linked to the computer by a platform-specific, plug-in interface board. Low level drivers allow the computer operating system to talk to the I/F boards.

Instrument drivers (e.g. within LabVIEW) allow to address the relevant features of each instrument, seen as one device, e.g. through a replica of the front panel. The user can choose to access a high level function, or get up to a particular register of the instrument.

The managing program (e.g. LabVIEW) must supervise the instruments setting and calibration, the measure sequence, data dependent decisions, the time scale, provide interaction with the user, authorisation protections, etc.

Data analysis, data presentation and storage, data transmission, are not mandatory parts of an instrumentation control software. However they are part of a well organized data acquisition system (e.g. they are part of LabVIEW), even better if they are integrated with the other types of operation.

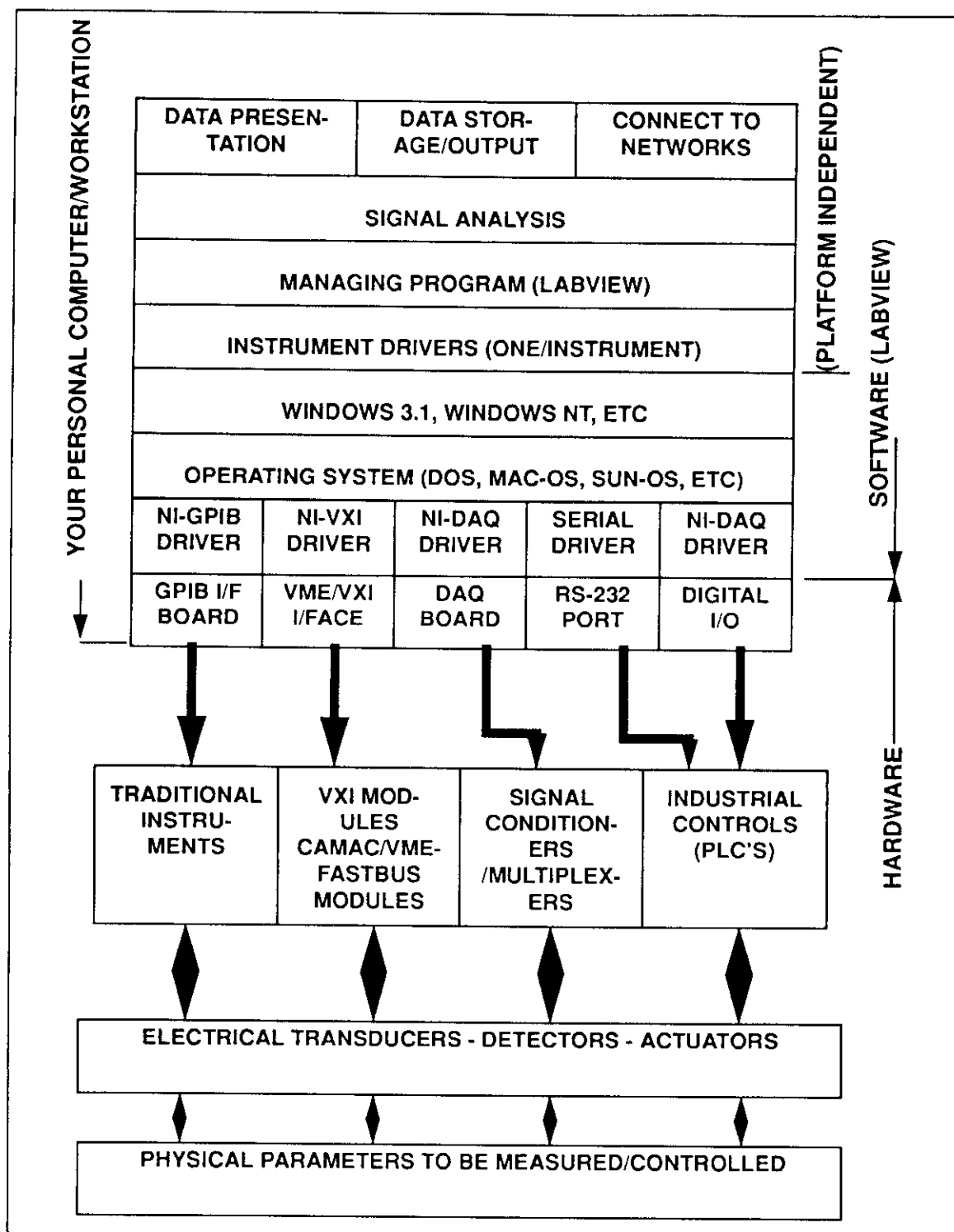


Figure 4.1 Structure of Software for Instrumentation Control

5 - THE IDEAL SOFTWARE

Looking back at Fig. 4.1, we could do a little exercise: imagine to have a measuring task, knowing more or less the instruments that we are going to use, and list on paper our definition of the "ideal software for instrumentation control".

So ideal, for instance, that if we receive all our instruments and materials today, and we have our Macintosh, PC or Workstation available, we can endeavour to have the whole thing configured and running, controlled at the screen of our computer, by tomorrow.

According to my list, the software should be:

Comprehensive

Using different types of software for the same application can cost development time. The ideal software should include all the functions shown in Fig. 4.1, viz. the low level drivers for the various categories of instruments (GPIB, VXI, Camac, VME, DAQ boards and communication ports), for all the operating systems; the high level (instrument) drivers; and provide all the functions needed to bring data through the analysis, presentation and storage level.

Easy to learn

to use, and to program. At its best, it should be a software for non programmers. Time spent in learning syntactical subtleties, metaphors, compiling and linking commands, etc. is subtracted from the real job of measuring and interpreting data.

GUI based

i.e. use a graphical user interface intuitive in operation, simple to apply, customizable.

Homogeneous

using the same programming method and user interface at all levels.

Modular

It is necessary to split complex programs into any number of modules, connected either vertically (hierarchically) or horizontally. Time is gained by developing each module independently from the others, and sharing it between different users and applications.

Portable

The software should be the same on every platform (PC, Mac, Sun, others). Better, every application developed should be portable w/o modifications.

Easy to debug

Obviously

Interactive

Obviously

Easy to maintain/modify

Only if a program is self documenting, it allows modifications by any user, at any time, and any stage (instrument configuration, data acquisition program, presentation).

Fast

PC's and alike are slow enough, when used with instruments, to tolerate also slow programs.

Powerful

A program which limits the size, type or speed of applications becomes quickly useless.

Cheap

It makes a difference whether it costs 2000\$ or 30000\$ (real cases, without names).

Stable

developments and collected data must last for years. So should do a program. Upwards compatibility is essential.

Standard

be vendor independent, and possibly labelled with an international norm.

Having thus defined the ideal software for our application, we can better appreciate the fact that a decade ago some people, in a then small company (National Instruments), sat down to do the same exercise. The difference is that they were enough visionary to start a project, and keep it alive for the long time that it took to bring it to completion. The story of LabVIEW development is interesting in itself, and worth telling.

Since the end of 1993, anyway, the program fulfils all the requirements that we have ideally listed, except one - it's a proprietary product.

Truly, the low level drivers exist only for the materials proposed by the company (butter on their bread), and some commercial instruments miss their drivers. But overall, the goal is attained.

6 - THE DEVELOPMENT OF LABVIEW

Most programs for the first generation of PC-controlled instruments (via GPIB bus), were written in BASIC.

BASIC had a simple and reusable set of commands, and interactive capabilities, but like any other text based programming language it required scientists, engineers and technicians to become programmers, i.e. to translate the knowledge of their applications into long and tedious line programs, even for simple measuring tasks.

Users with little or no programming experience, or occasional users were discouraged by arcane syntax and messages¹. As a result, data acquisition was automated only when it was absolutely necessary.

Nat.Inst. which had its own team of programmers struggling to develop tools for the control of instrumentation, was sensitive to the programming burden placed on engineers.

In April 1983 Jeff Kodosky and other cofounders of Nat.Inst. decided that only a new tool for developing instrument software would make the computer accessible to non-programming users.

But what form a new language should take? The original ideas were:

- a) A graphic user interface. Most engineers learn about an instrument by studying its front panel and experimenting with it. Hence the most familiar and intuitive user interface had to be a facsimile of a real instrument front panel.
- b) A graphic programming method. Designers who start a project, draw as the first thing a block diagram. It helps to visualize the problem and suggests a design. Why not translating it into a design tool, using labelled icons to represent the functions, and electric-like wires to symbolize data transmission. A data-flow-based operation was chosen, to simulate real systems, and some programming structures like loops and iterations were added in a special graphic form. Hence the name of "structured data flow programming".

With these ideas in mind, J.K. and a few others hired a 80 sqm loft, away from the company but near the university of Austin; they brought in ten small Macintosh (512K memory) chosen for their graphics capabilities, 10 young people out of school, a refrigerator and a microwave oven. There were no windows and

1. Remember? (beep) SYNTAX ERROR AT LINE 3457

no visible clock; work would most often go on overnight, interrupted by sudden, collective discussions on interesting topics.

The early development went on very quickly, but getting to a viable product was less easy. The task had been largely underestimated, the machines available had many limitations, etc. When, three years later, they had to release version 1.0 for Macintosh, the program was slow even compared to interpreted Basic, and defective.

In 1987 came version 1.2; now the program was reliable and robust, proving the original idea good, but still too slow for large applications. The only solution was to make it compiled, but that unfortunately required a complete redesign!

In 1990 LabVIEW was fully rewritten, taking also into account many user remarks. When version 2.0 was shipped, it was fast, it incorporated Object Oriented Programming and, as a new feature, an almost instantaneous integral compiler. For the user the compiled version was as flexible and interactive as the previous interpreted version.

It was running only on Macintosh, a platform unfamiliar to engineers and originally intended only for prototyping. At that time, PCs and DOS still lacked the graphics capability and the 32-bit addressing support necessary to a large, sophisticated application; Windows 3 had still to come.

By luck, the Mac II had an open architecture, and so Nat.Inst. started to produce plug-in boards for the Mac + LabVIEW association, while other companies were mostly producing plug-in boards for PC's.

In the following years, with the appearance of new operating systems, Nat.Inst. decided to give LabVIEW a new portable architecture, and to produce LabVIEW for other platforms.

In Aug. 1992 LabVIEW ver. 2.5 would run also on PC and Sun platforms. Once more, most of the machine dependent ('manager') layer had been redesigned. Although they would appear the same program to a Macintosh or a PC user, the two versions were not compatible.

Only at the end of 1993 (ten years later!), with version 3.0, the architecture was unified, according to the original idea of the founders. The applications (VI's) are fully portable between platforms - and such will hopefully remain.

7 - LABVIEW FEATURES

"The user's time is more valuable than the developers' time". This axiom, today conventional wisdom, has taken a few years (and Windows 3) to be accepted. New, user-friendly programming tools which reduce the effort of individual software developers are the answer. Software modularity, maintainability and reusability, intrinsic in LabVIEW hierarchical and homogeneous structure, are vital in controlling the cost of test programs of increasing complexity.

By working with a few LabVIEW examples we will examine the LabVIEW features that go in the direction of facilitating both the developer's and the user's task, and which make LabVIEW the ideal software for instrumentation control.

The features that we want to point out are the following:

- graphic user interface
- graphic programming method, self documenting
- dataflow execution
- integral compiler
- modularity and homogeneity
- instrument drivers
- data analysis libraries
- data presentation libraries
- communication/storage tools
- inter-platform portability

Graphic User Interface (GUI)

Graphic interaction is generally preferred to textual interaction. The LabVIEW built-in graphic user interface is the same for all levels of hierarchy; it's intuitive in operation, simple to apply and to customize, and nice to look at. Control panels that mimic real panels are adapted also to unskilled users.

VI's

The concept of virtual instrument (VI) has been pioneered by LabVIEW; it allows for instance to transform a real instrument (e.g. a voltmeter) into another, software based instrument (e.g. a chart recorder), thus increasing the versatility of available hardware. The VI concept is so fundamental to the way that

LabVIEW works, that programs are in fact called VIs. Virtual Instruments are particularly useful when using panel-less instruments, like high performance ADC plug-in boards or VXI modules. All VI's follow the same rules, whether they describe low level or top level functions, and can be modified, interchanged and combined. In addition, VI's are easy to understand and to maintain.

Graphical Programming

LabVIEW programming is done via a block diagram, consisting of icons and wires, which directly couple to executable code; there is no underlying procedural language or menu-driven system.

LabVIEW iconic programming is not simply an organizational tool - it's a true programming language complete with multiple data types, programming structures and a sophisticated compiler.

The LabVIEW programming structures are the same as in classic, textual languages: For loops, While loops, Sequence and Case instructions, with all the corresponding tools. Icons are placed within structures in the same way as code lines are enclosed in a textual structure.

High level operations are encapsulated in convenient VI libraries, either supplied with LabVIEW, or developed by users.

The precise method of producing VIs, and their self documenting character, favour the development of large projects.

Dataflow Execution

LabVIEW programming is based on modern Object Oriented programming, and data flow execution. With data flow programming, an object cannot be executed until all its input data are present. The same way, the object outputs cannot be used until the object has executed. It is the data flow between connected objects which controls the execution. This method makes free from the linear (sequential) structure of textual languages. In LabVIEW several paths can be created and executed simultaneously.

Fast Compiler

LabVIEW includes a graphic compiler which generates a machine-optimized code, executable at a speed comparable to that of C-compiled programs. From the user's point of view, the compiler is fast and invisible, and the programs are as interactive as interpreted ones. One can also create applications for the Run-time version of LabVIEW or as executables, with the application builder.

Modularity

Complex programs must be subdivided into an arbitrary number of modules, connected either horizontally or vertically (hierarchy). Every module must be testable/developable independently, and shareable between users and applications.

LabVIEW is unique for hierarchy, expandability and homogeneity. It also has the advantage of using the same programming method (and user interface) for any type of function (low or high level).

Every Virtual Instrument (VI) can serve as a sub-program for another VI, and sub-VI's can be opened interactively to examine their way of working.

Instrument Drivers

LabVIEW includes the low level (interface) drivers for four types of data acquisition material - plug-in I/O boards - GPIB controlled instruments - VME/VXI modular instruments - RS232 instruments. Message-based (GPIB, RS232, VXI) instruments can be mixed with register-based (VXI, DAQ boards) instruments without conflicts.

Drivers are supplied with libraries of functions for the hardware programming; the options they offer are independent of the platform, of the operating system, often of the instrument. In addition to the low level drivers, there are high level (instrument) drives for the most popular types of instruments.

The source code of instrument drivers is modular, readable and modifiable according to the developer's taste.
Re-configurable instruments are more flexible.

Data Analysis

The real benefit of virtual instruments, and the power of LabVIEW, appears when acquired have to be fed into the analysis VIs.

No special data formatting, or use of other proprietary software is needed, as data flow smoothly between the various applications.

LabVIEW contains more than 170 analysis functions. These functions fall into several major groups: signal generation/simulation, DSP, digital filters, smoothing windows, statistics, array operations, integral/derivative, and pulse/threshold detection.

Data Presentation

Data presentation is the final element of a complete data acquisition, it's an often neglected, yet time consuming task for programmers. Many data acquisition system are only as good as their data presentation.

In the past, data were presented through text based interfaces,

sometimes obscure, unsuitable to demonstrate the relational information of data.

LabVIEW offers simple routines to display and correlate data graphically, and thus maximize the transmitted information.

Communication

LabVIEW has drivers for TCP/IP network connection and for connection to databases. Special drivers are available for Igor, Mathematica, Matlab or HiQ, supporting the file formats of these applications.

Portability

LabVIEW runs on the most popular platforms, Apple Macintosh and Power PC, IBM PC compatibles, Sun Sparcstations and DEC stations.

The user interface and all the library functions are the same on each platform, and the programs (VIs) are portable without modification.

8 - CONCLUSION

LabVIEW is a complete tool, and a complete program. It allows the user to perform from the lowest level functions, like accessing a particular register of an instrument, to the highest level ones, like displaying the result of advanced spectral analysis, without having to resort to the use of another programming language. It has the computational ability of a classical programming language and the parallelism of concurrent programs.

It is also an excellent automation tool. Note the difference between simple data acquisition, and automated, hands-off processes, in which the computer governs a sequence of events, takes measurements (and decisions) and presents the results. Many operations require automation: long term, low speed data collection such as environmental monitoring and control, high speed operations such as pulse power diagnostics, collecting lot of data in a short time, repetitive operations such as test and calibration of series produced equipment, high precision operations beyond human capability, such as star tracking with a telescope, complex processes with many simultaneous inputs outputs

LabVIEW is suitable for all these tasks.

Finally, programming with LabVIEW is fun! (anyway, more than C).

9 - ANNEX

REAL TIME

The precise definition of real time depends on your system temporal response. This includes both the cycle time or sampling rate for data exchange and the computer system's latency - the time it takes to respond to an external occurrence.

For example, the real time control of a house heating system requires a response time of about 1 minute (even PCs can do that!). The real time flight control of an F16 plane are sampled at 45Hz, and so on, all the way down to nanosecond phenomena.

For LabVIEW the real time scale unit is about 0.1 sec (if you avoid receiving mail on a Sun, or inserting a floppy in a MAC, or hold down the mouse button, or scroll down a menu).

For more stringent needs it's better to use a dedicated machine or an external smart controller (PLCs, embedded processors).

MACHINE REQUIREMENTS

The minimum memory for LabVIEW use is 8 MBy, and 20-32MBy for professional applications.

Similarly, the minimum Hard Disk space is 120 MBy, but 250-500MBy for professional use.

IMITATORS

Following LabVIEW, most instrument softwares offer a front panel oriented user interface. Quite often it is simply a display/top level user interface added to an otherwise conventional, textual language based programming system.

Another imitated feature is iconic programming - but often icons hide just conventional programming methodologies. These icons offer a pre-set number of operations, simple to use because of their turnkey nature, but impossible to expand/modify without reverting to standard programming languages.

LabVIEW iconic programming is not simply an organisational tool - it's a true programming method, as we have seen.

10 - Slides

Instrumentation

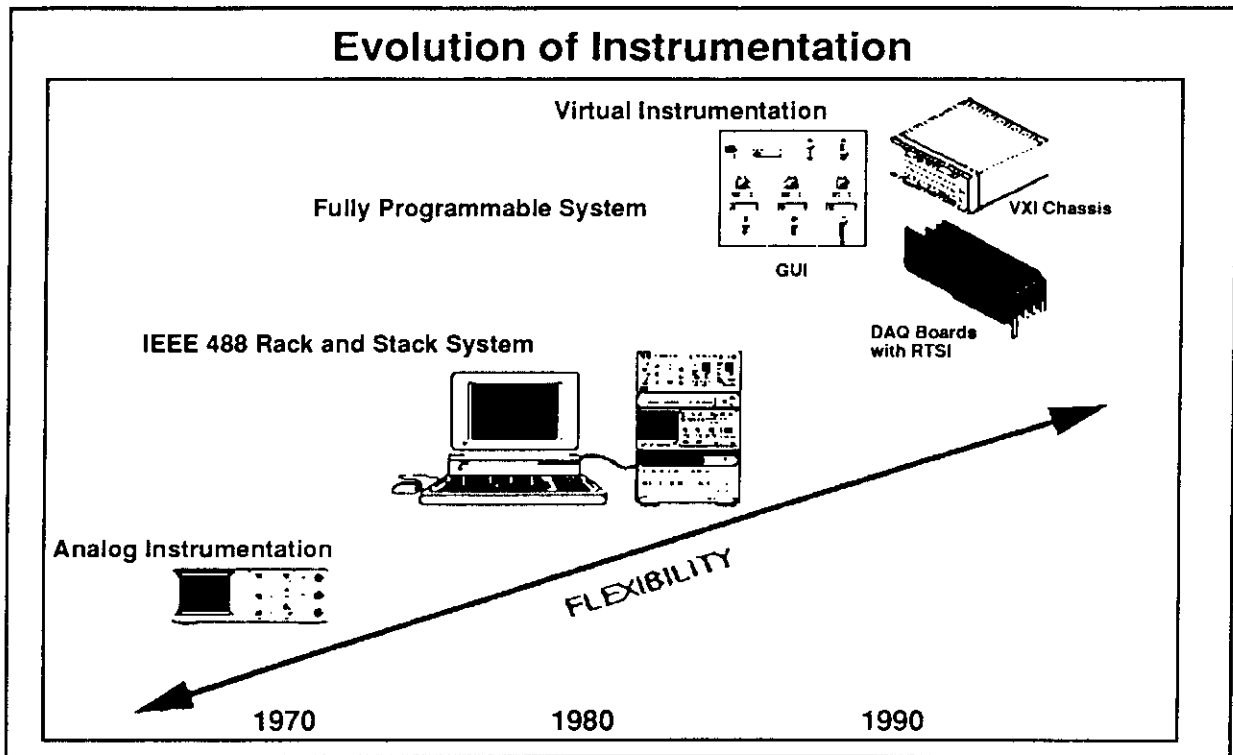
- Slide 1a - Evolution of Instrumentation
- Slide 1b - Instrumentation Today
- Slide 2a - GPIB Programming Example
- Slide 2b - RS-232 Serial Connection
- Slide 3a - Plug-In Data Acquisition Boards
- Slide 3b - Multifunction I/O Board
- Slide 4a - Wide Range of Performance
- Slide 4b - Analog Signals
- Slide 5a - SCXI - Three-Port Signal Conditioning System
- Slide 5b - VXI "VME Extensions for Instrumentation"
- Slide 6a - VXI - An Open Architecture with Shared Processor Bus and Timing
- Slide 6b - VXI Controllers
- Slide 7a - VXI Instrument Drivers
- Slide 7b - Ethernet based Control System
- Slide 8a - PCMCIA GPIB Interface
- Slide 8b - Software for Instrumentation

Data Acquisition

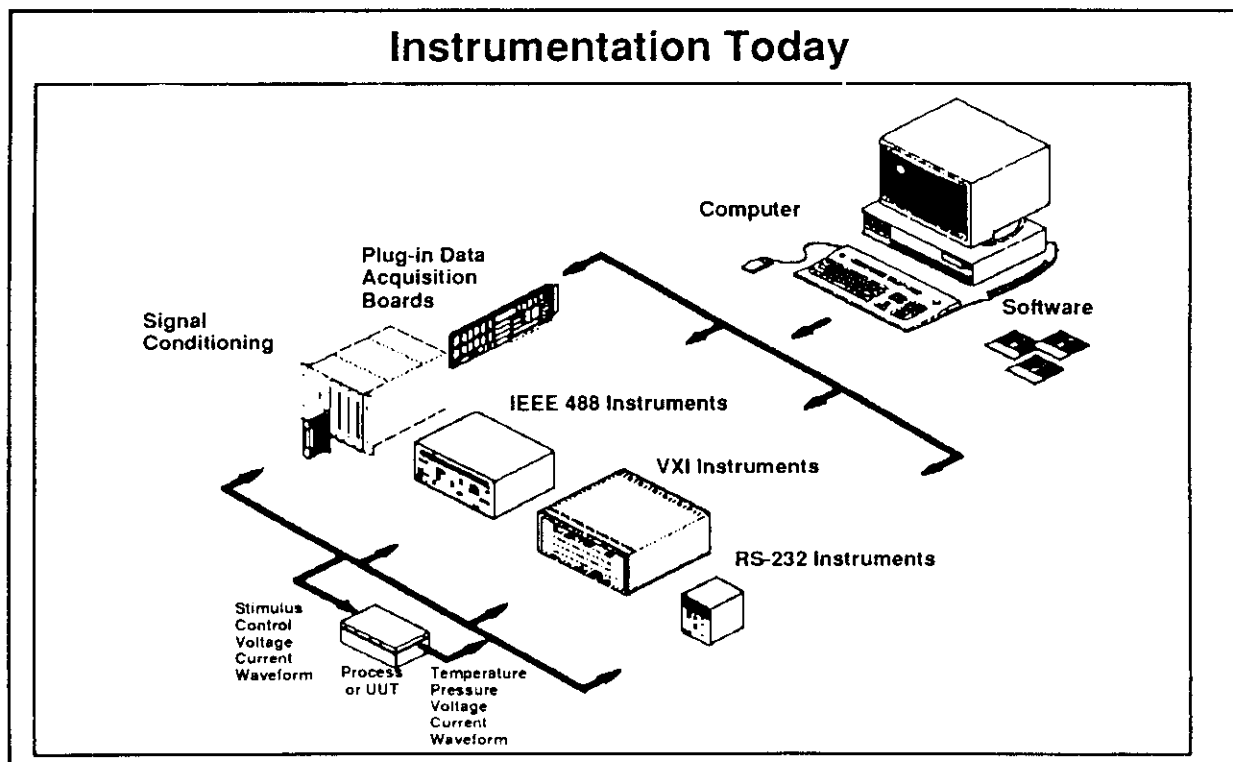
- Slide 9a - The Ideal System
- Slide 9b - The LabVIEW Lego
- Slide 10a - Virtual Instrument Framework Example - LabVIEW
- Slide 10b - LabVIEW - A GUI and a Graphical Programming Method
- Slide 11a - LabVIEW Product History
- Slide 11b - Graphical Programming

Data Analysis

- Slide 12a - Analysis in LabVIEW
- Slide 12b - Digital Signal Processing (DSP)
- Slide 13a - Smoothing Windows
- Slide 13b - Joint Time-Frequency Analysis
- Slide 14a - Gabor Spectrogram
- Slide 14b - Digital Filters
- Slide 15a - Time Domain Analysis
- Slide 15b - Time Domain Analysis
- Slide 16a - Signal Generation/Simulation
- Slide 16b - Other Analysis Functions
- Slide 17a - Analysis Summary
- Slide 17b - Data Presentation Options
- Slide 18a - Screen I/O
- Slide 18b - File I/O and Hard Copy
- Slide 19a - Dynamic Data Exchange
- Slide 19b - Networking in T&M
- Slide 20a - Integrating your Instrumentation

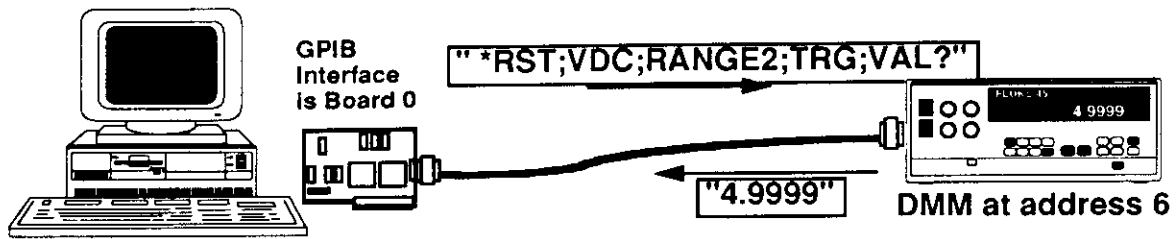


Slide 1 a)



Slide 1 b)

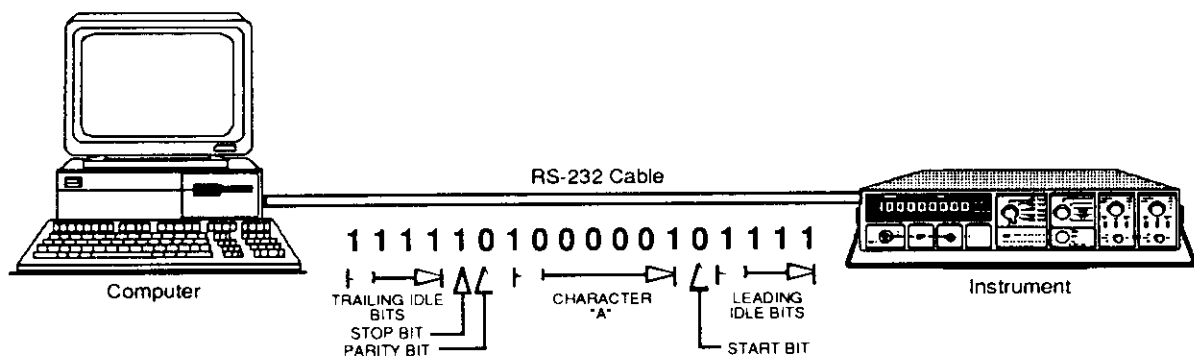
GPIB Programming Example



- Remote programming standard since 1975
- 8-bit parallel protocol
- Transfer rate of more than 1 Mbytes/s
- Standard cable
- Control up to 14 instruments
- Large installed base of instruments

Slide 2 a)

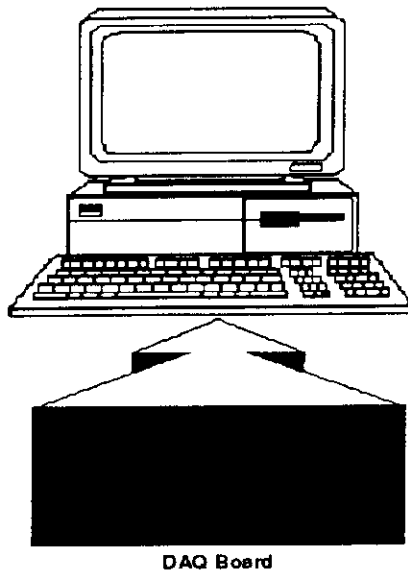
RS-232




- Port is standard on most computers; cabling is not standard
- Each port can communicate with only one instrument
- Common RS-232 instruments – digital thermometers, scales

Slide 2 b)

Plug-In Data Acquisition Boards



Analog
Input/Output



Digital
Input/Output



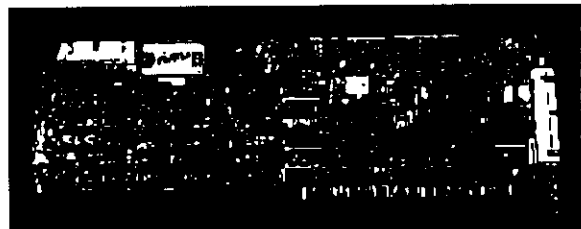
Timing
Input/Output



Slide 3 a)

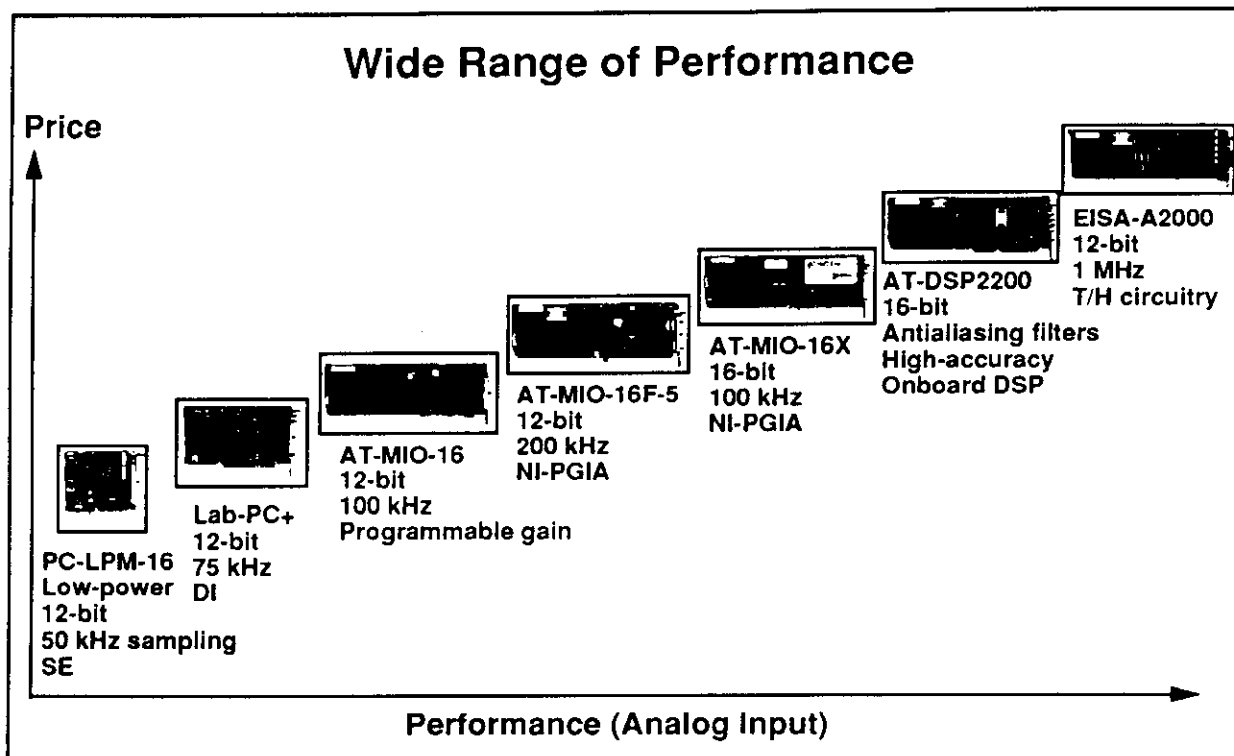
Multifunction I/O Board

- 16-bit ADC with 16 analog inputs
- Programmable gains up to 100
- 100 kHz sampling rate at all gains using NI-PGIA
- 256-sample deep FIFO
- Pretrigger and posttrigger modes
- Continuous and interval scanning
- Two 12-bit DACs
- Eight digital I/O lines
- Three 16-bit counters
- Self-calibration

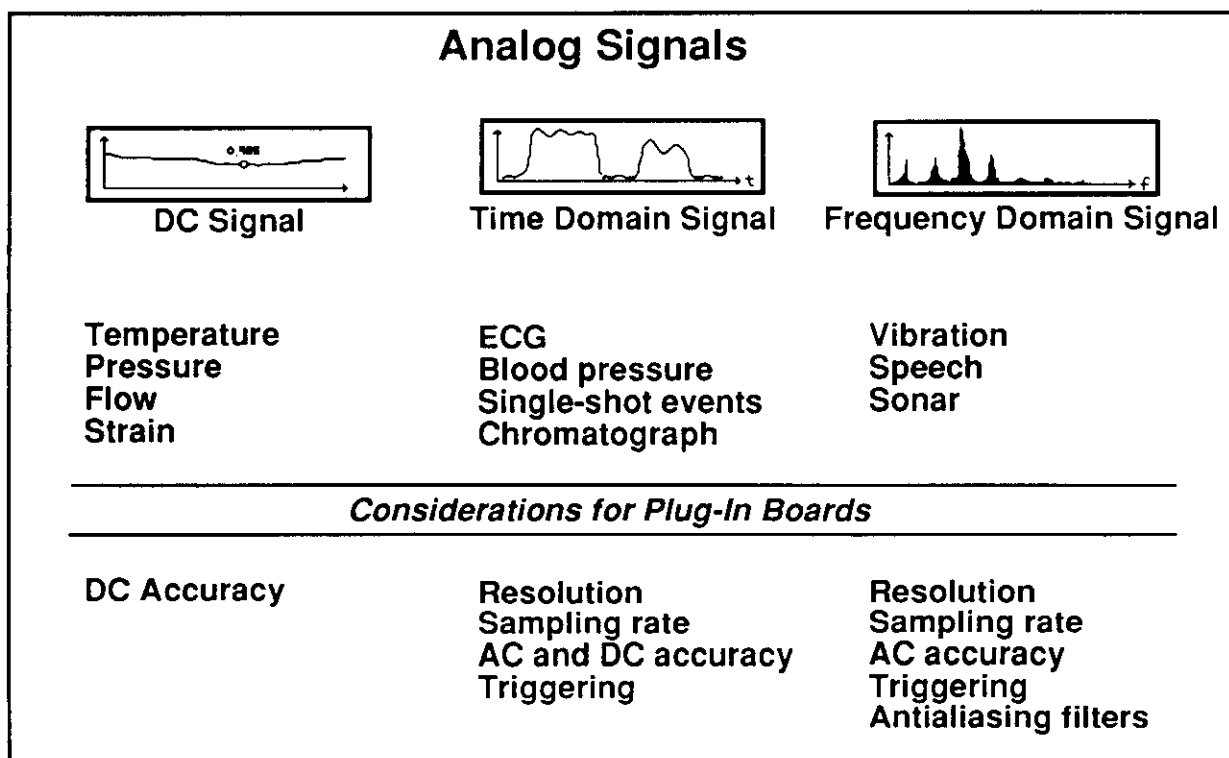


AT-MIO-16X

Slide 3 b)



Slide 4 a)

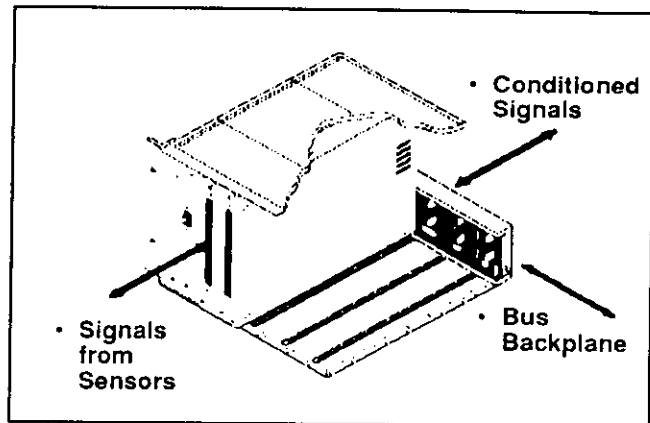


Slide 4 b)

SCXI – Three-Port Signal Conditioning System

Signal Conditioning

- Multiplexing
- Amplification
- Isolation
- Filtering
- Excitation
- Cold-junction compensation
- Signal and device switching

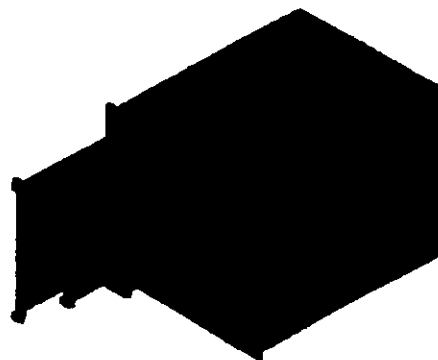


Slide 5 a)



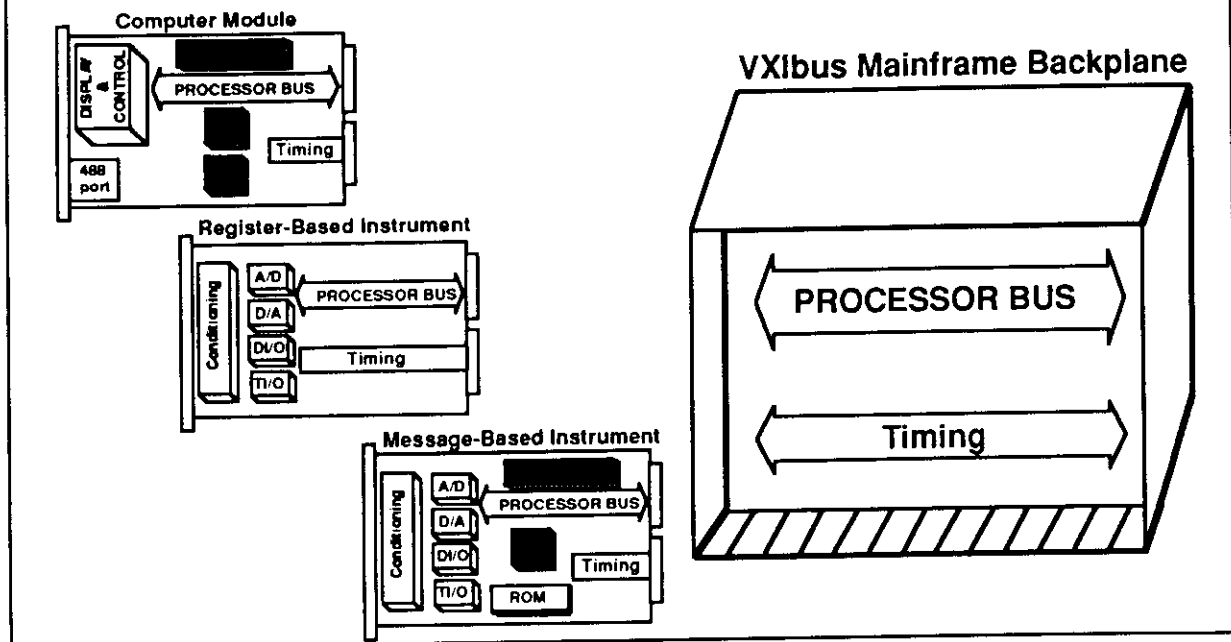
VXI "VME Extensions for Instrumentation"

- IEEE standard 1155
- Combines instruments with modern computer architecture
- More than 50 vendors offering VXI products
- More than 500 VXI instruments
- Lacks front panels – requires GUI



Slide 5 b)

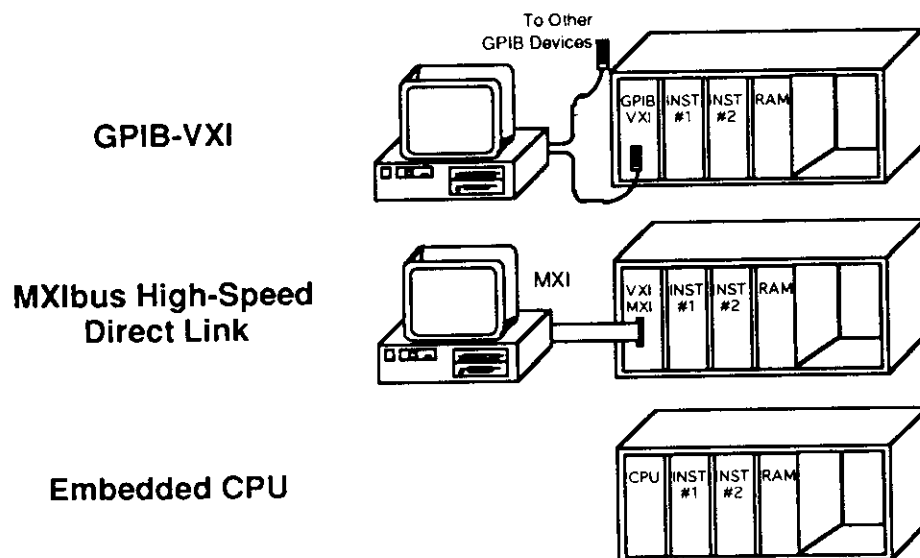
VXI – An Open Architecture with Shared Processor Bus and Timing



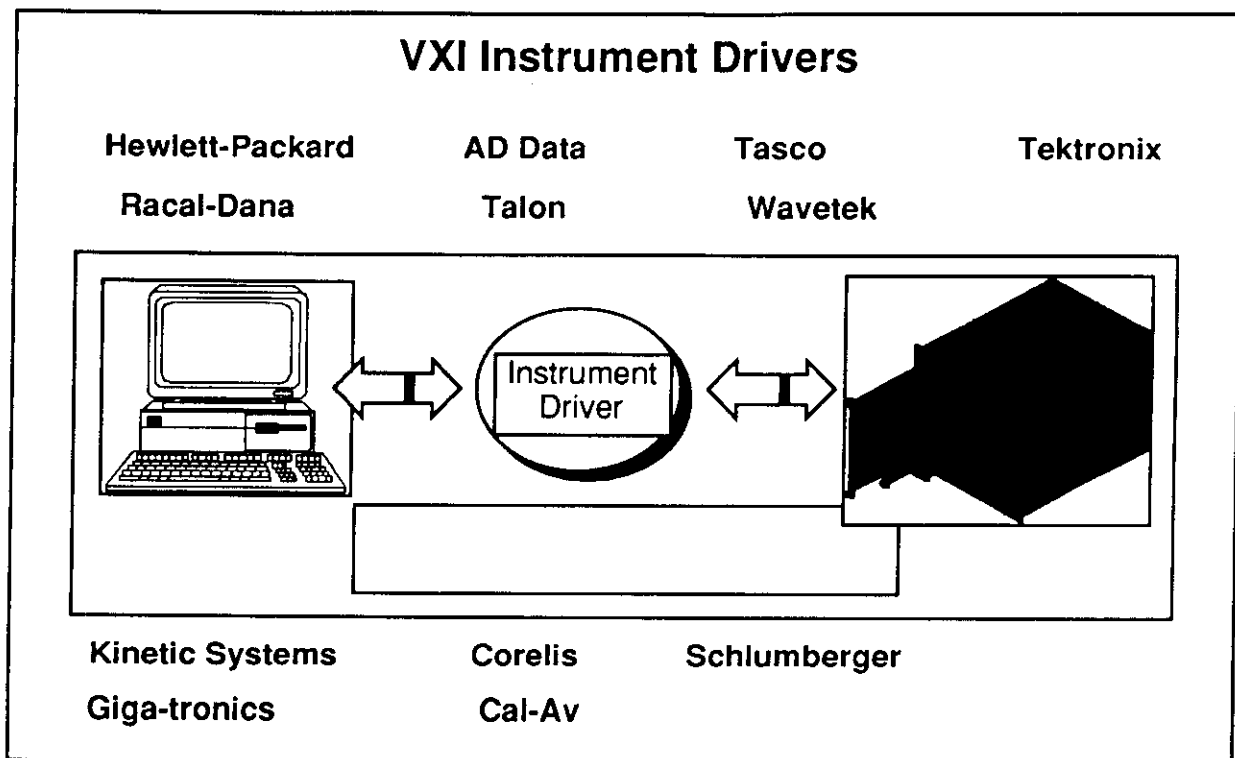
Slide 6 a)



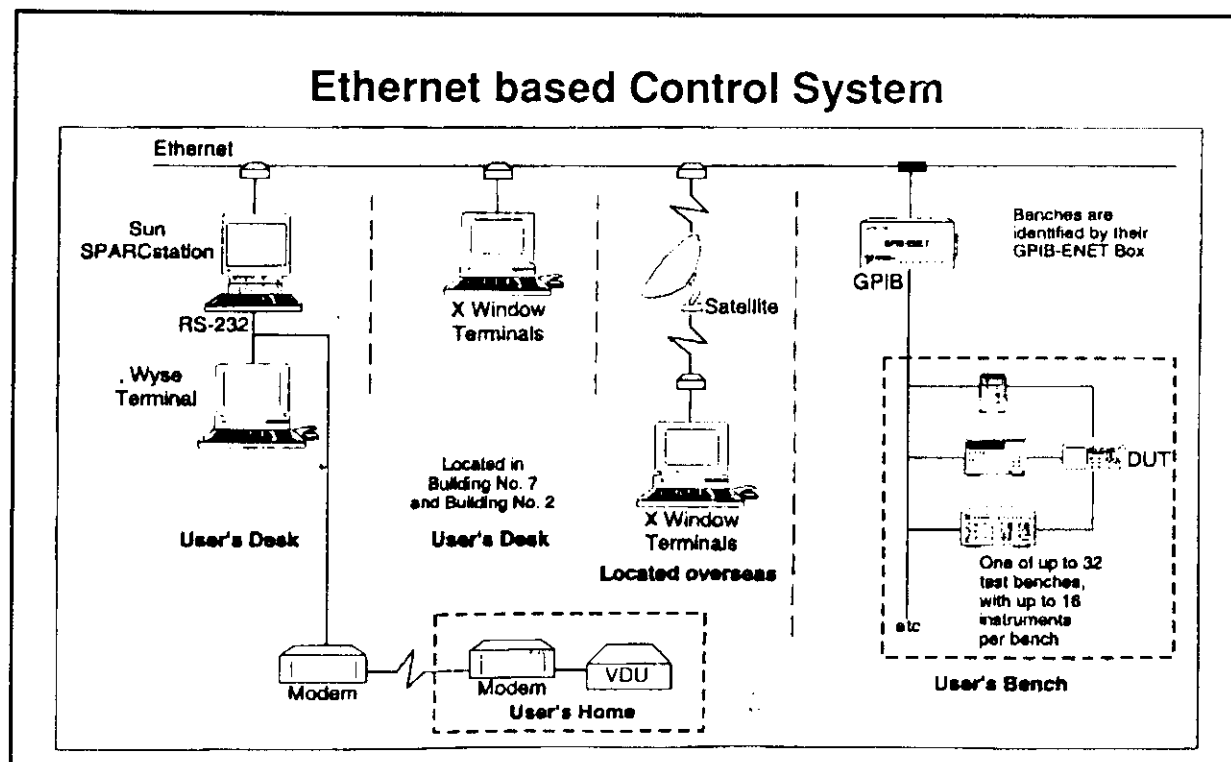
VXI Controllers



Slide 6 b)



Slide 7 a)

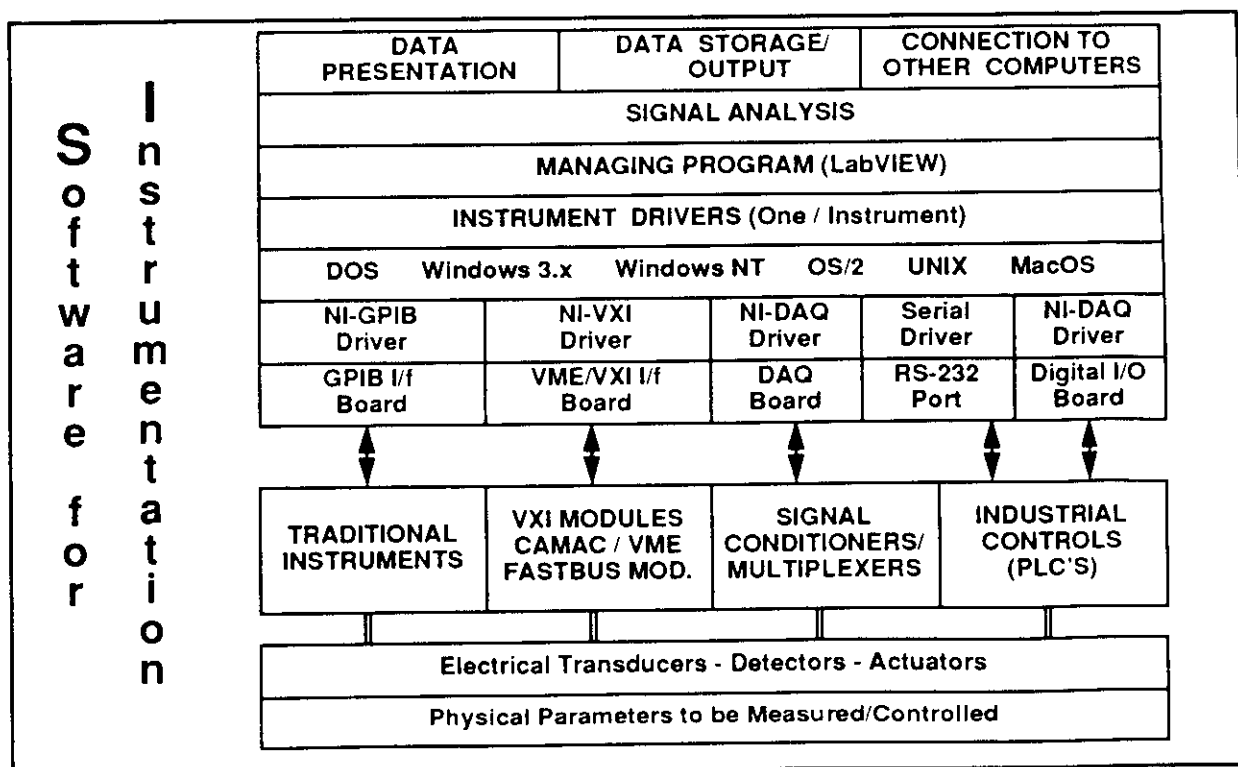


Slide 7 b)

PCMCIA GPIB Inter- face



Slide 8 a)

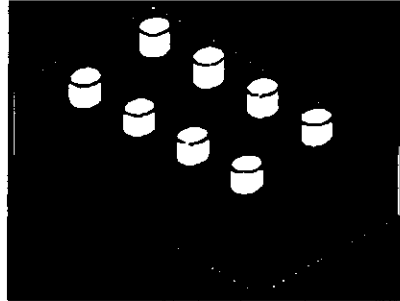


Slide 8 b)

THE IDEAL SYSTEM

The LEGO!

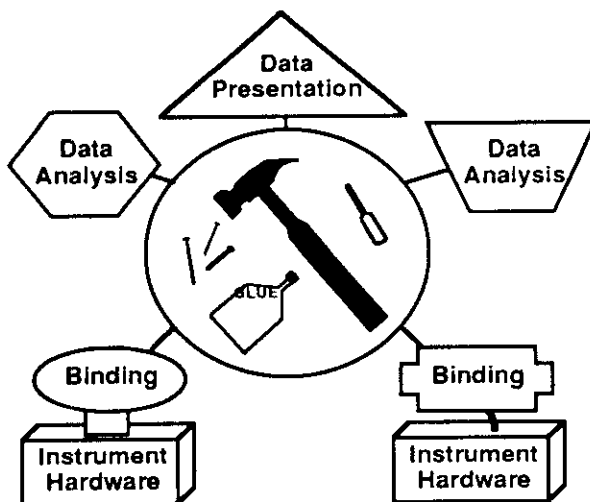
- Intuitive operation
- Structured design and packaging
- Modular, reusable components
- Consistent, unified paradigm
- Standard interface, top to bottom



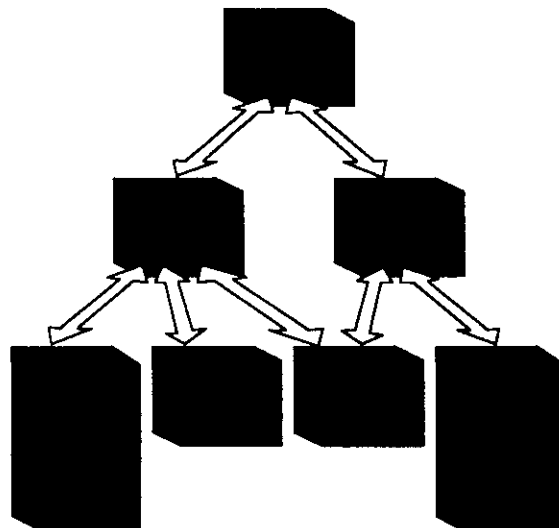
Slide 9 a)

The LabVIEW Lego

Non-Virtual Instrument Framework

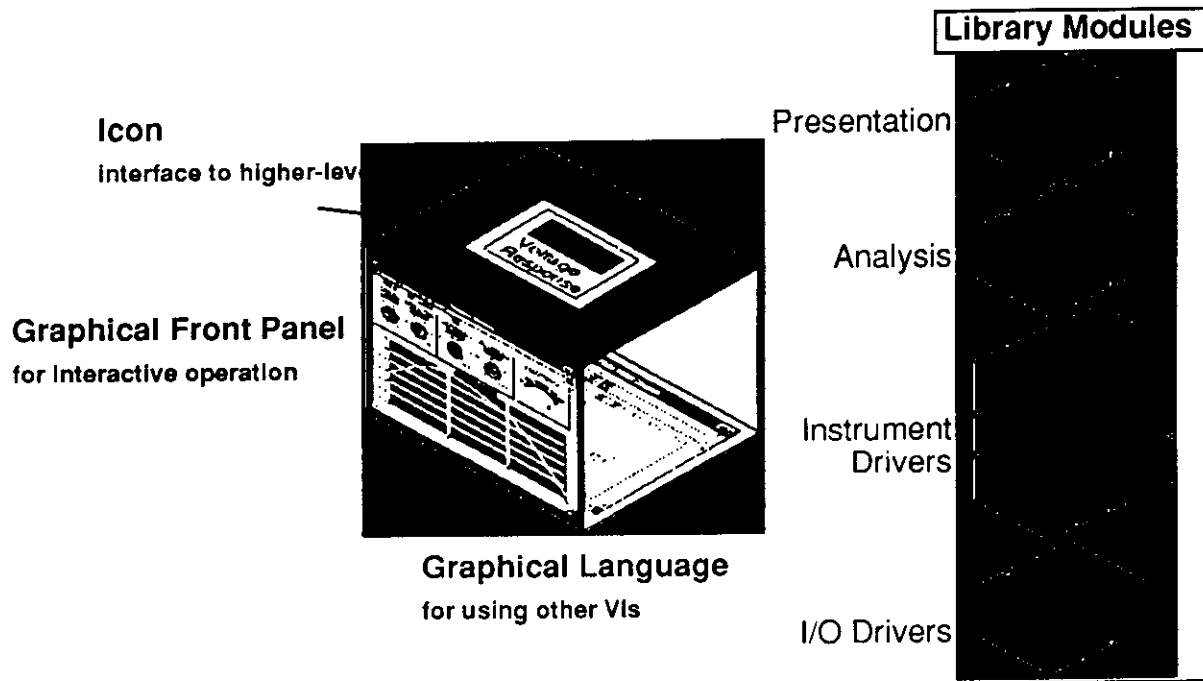


Virtual Instrument Framework



Slide 9 b)

Virtual Instrument Framework Example – LabVIEW



Slide 10 a)

LabVIEW - A GUI and a Graphical Programming Method

Purpose

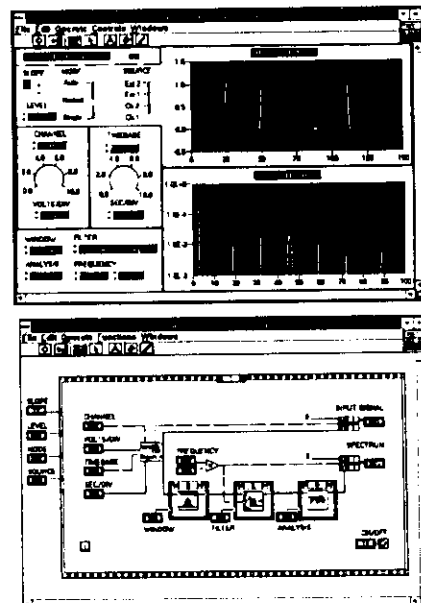
- Software to simplify development of automated measurement and instrumentation systems
- Virtual instrument concept

Front Panel

- Graphical user interface (GUI)
- Intuitive control of instrumentation

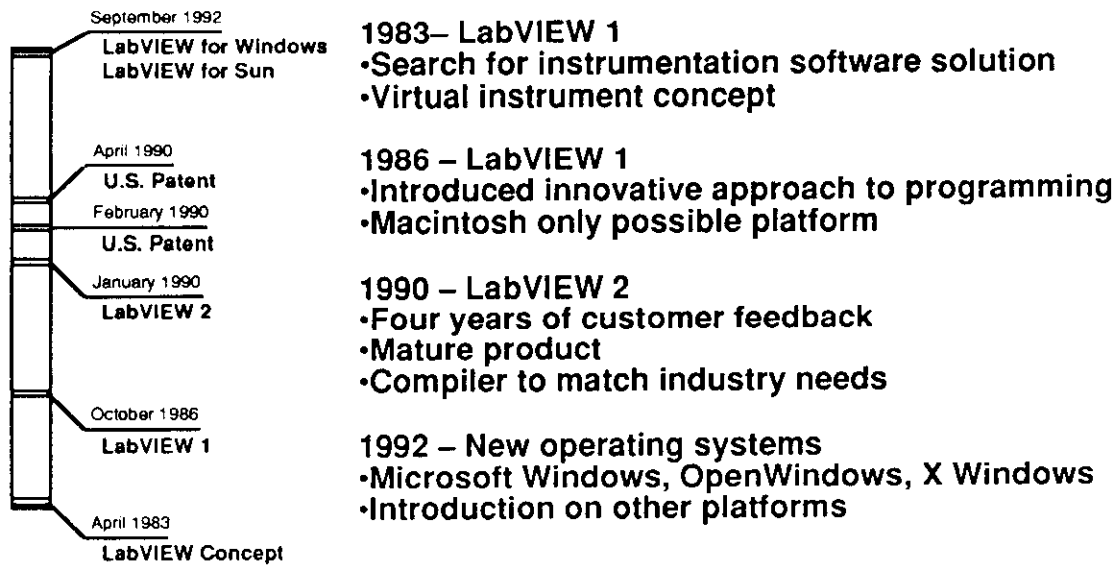
Block Diagram

- Familiar technique
- Rapidly build, test, and modify



Slide 10 b)

LabVIEW Product History



Slide 11 a)

Graphical Programming

Dataflow Programming

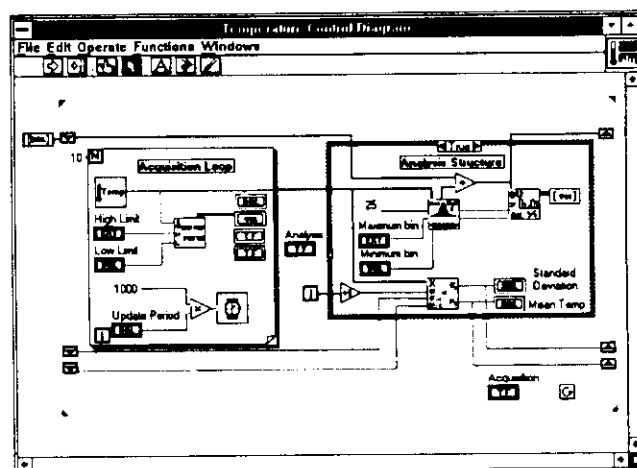
- Nonlinear structure
- Multitasking capabilities
- Programming structures

Hierarchy

- Modular design
- New building blocks
- Multilevel system

Graphical Compiler

- Execution speed
- Run-time system



Slide 11 b)

Analysis in LabVIEW

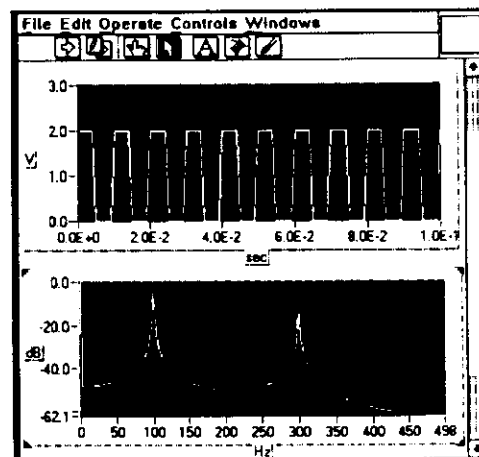


- Tightly coupled with acquisition VIs
- No need to format acquired data
- More than 170 functions

Slide 12 a)

Digital Signal Processing (DSP)

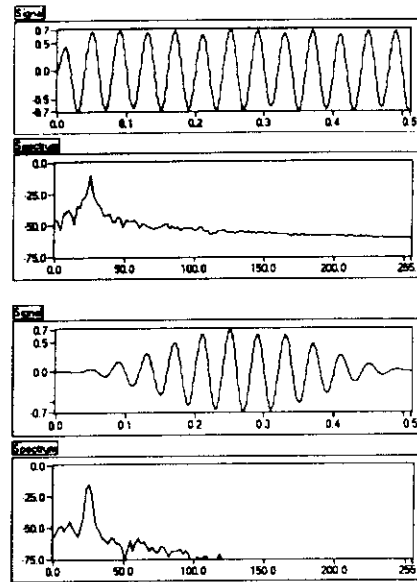
- Spectral analysis
- Conversion to frequency domain
 - Fast Fourier Transform (FFT)
 - Power spectrum



Slide 12 b)

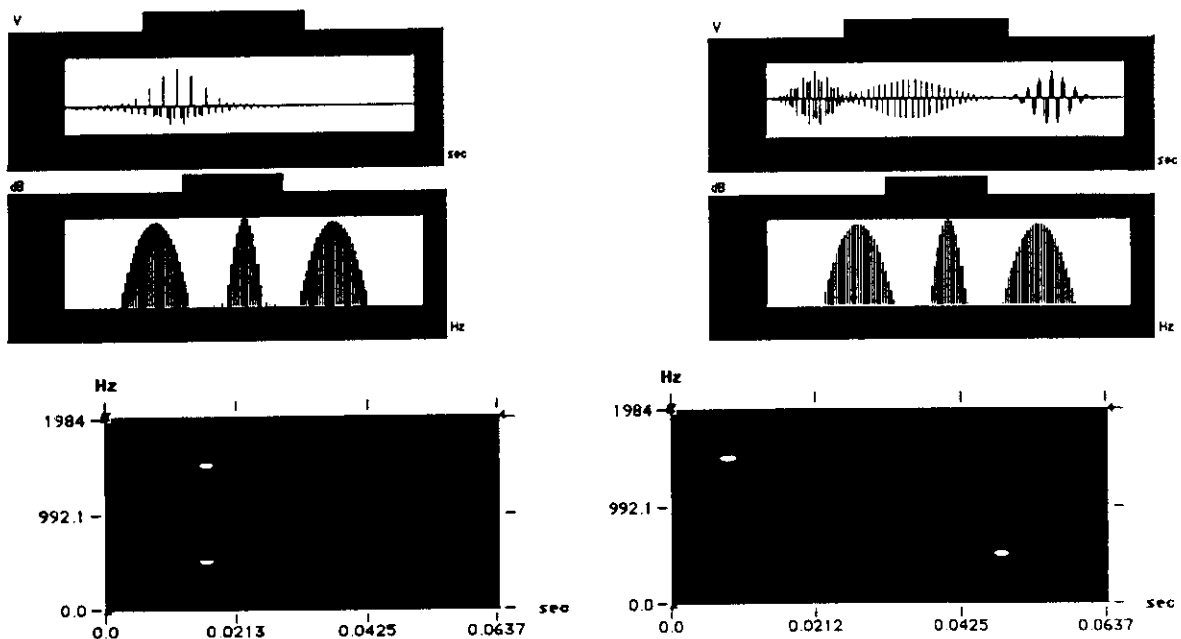
Smoothing Windows

- Used in conjunction with spectral analysis (FFTs)
- Minimize the effect of discrete sampling



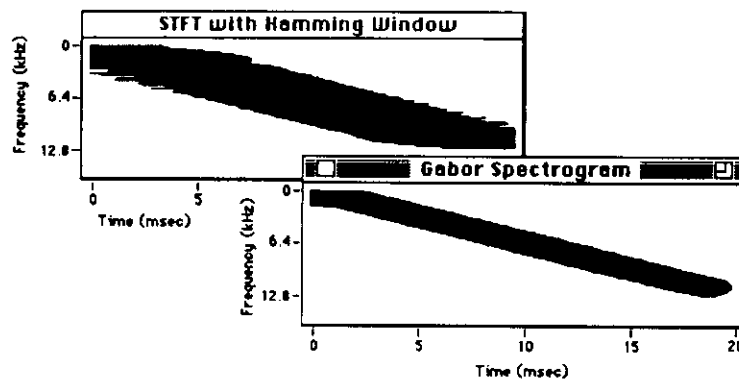
Slide 13 a)

Joint Time-Frequency Analysis



Slide 13 b)

Gabor Spectrogram

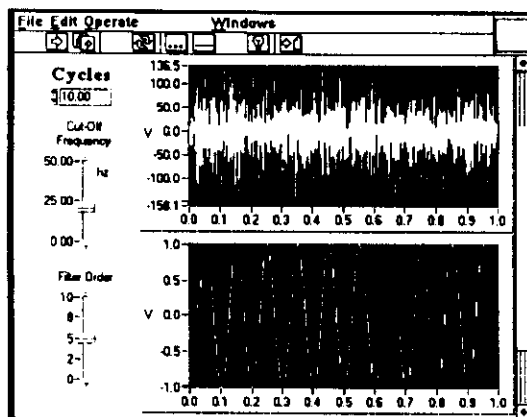


- Faster than STFT
- Better SNR
- Solves window selection problems

Slide 14 a)

Digital Filters

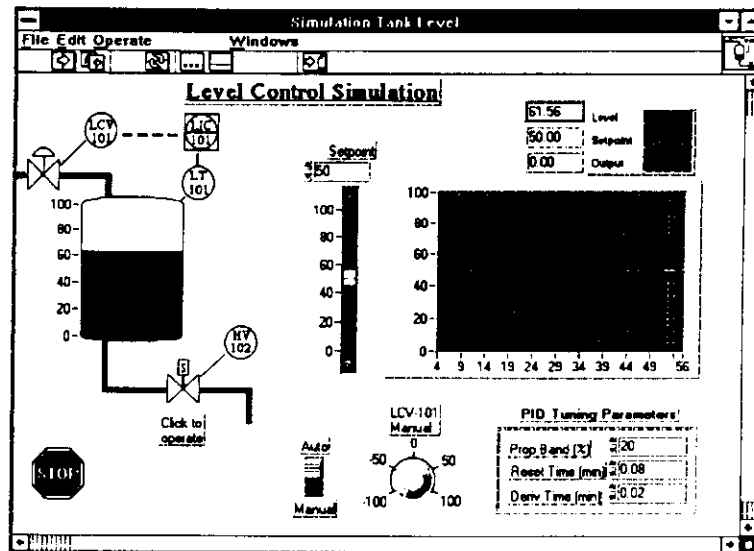
- Eliminate noise and undesired frequencies
- Superior to analog filters
- No drift – not affected by temperature or humidity
- Don't require precision components
- Cost effective



Slide 14 b)

Time Domain Analysis

- Integral
- Derivative

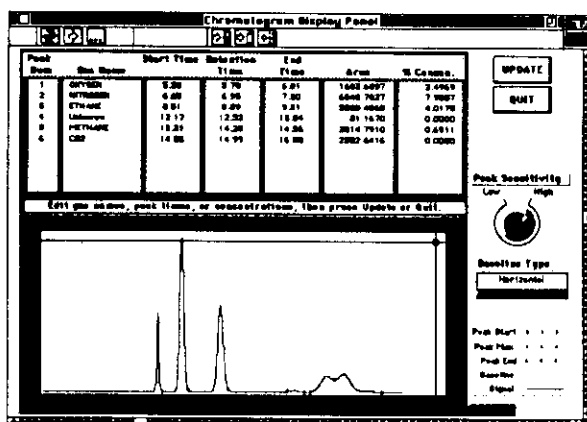


Process Control

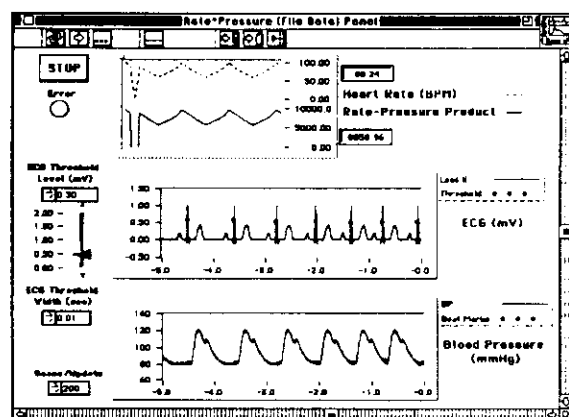
Slide 15 a)

Time Domain Analysis

- Peak detection
- Threshold detection



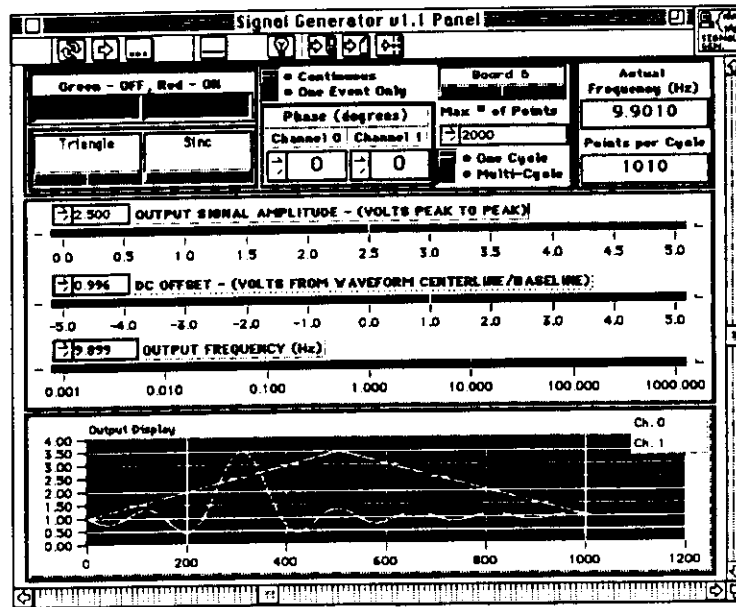
Chromatography



Biomedical Monitoring

Slide 15 b)

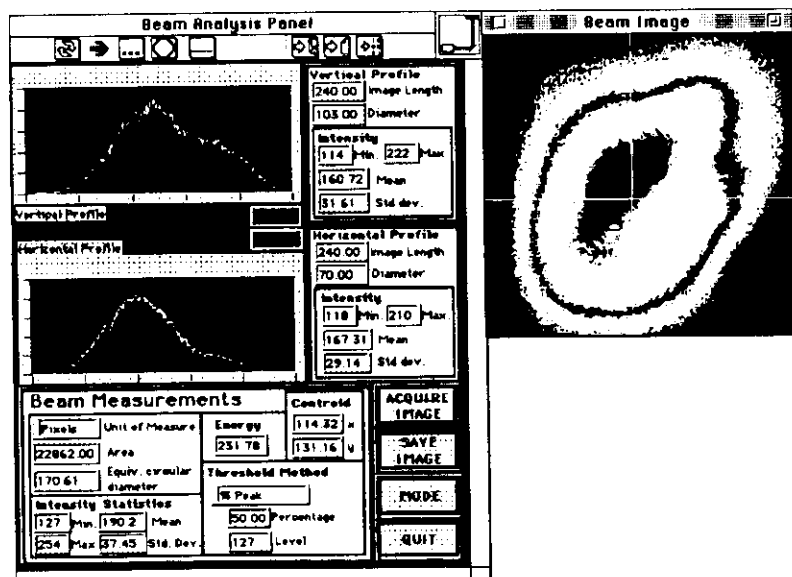
Signal Generation/Simulation



Slide 16 a)

Other Analysis Functions

- Statistics
- Numerical analysis
- Curve fitting



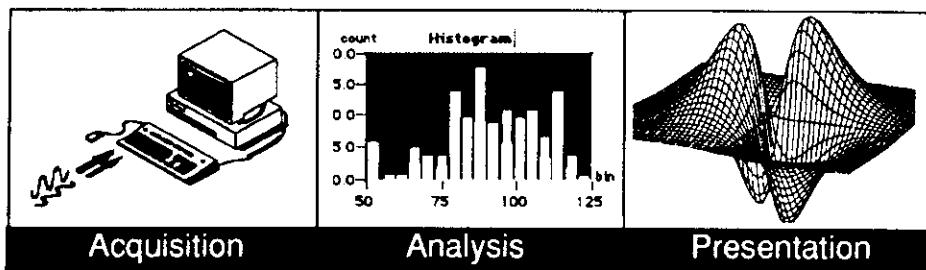
Slide 16 b)

Analysis Summary

- Key to building virtual instruments
- Define your own instruments
- Tightly coupled with acquisition VIs
- No need to format acquired data
- More than 170 functions
 - DSP
 - Smoothing windows
 - Joint time frequency analysis
 - Digital filters
 - Signal generation
 - Statistics
 - Numerical analysis
 - Curve fitting

Slide 17 a)

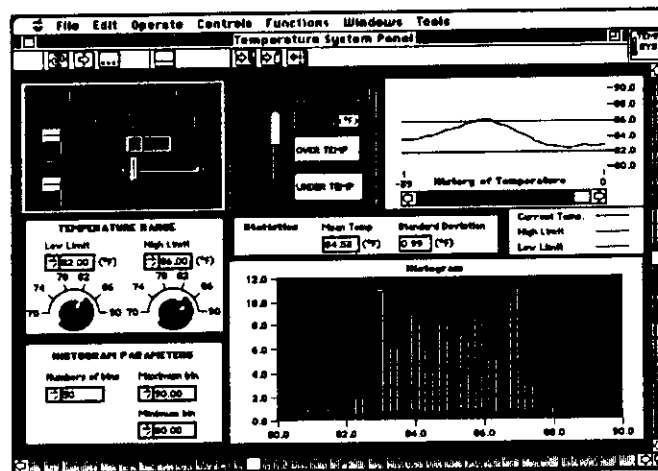
Data Presentation Options



Slide 17 b)

Screen I/O

- Text display
 - Cryptic
 - Sequential
- Graphical user interface
 - Intuitive
 - Interactive/nonsequential
 - Context sensitive
 - Needed by VXI
- GUI editor
 - Standard objects
 - Custom objects

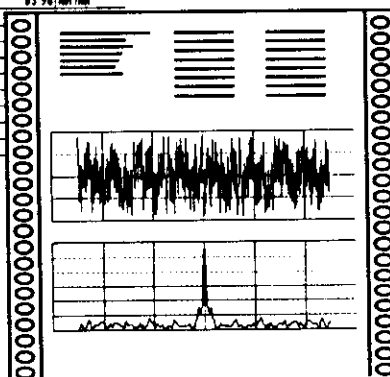


Slide 18 a)

File I/O and Hard Copy

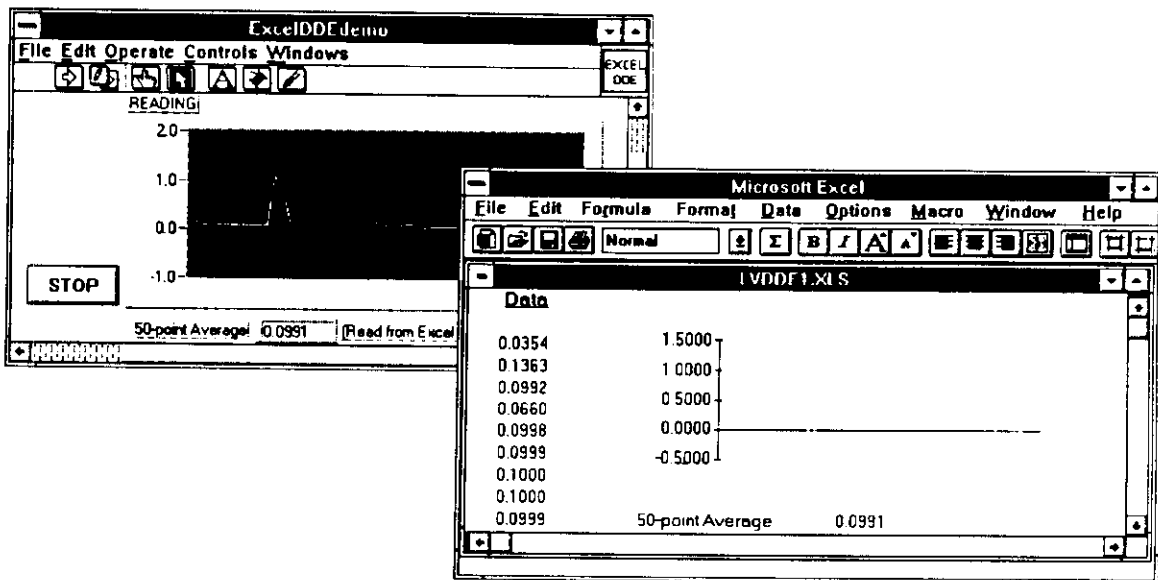
- File I/O
 - ASCII
 - Binary
 - Spreadsheet
- Hard copy
 - Text printers
 - Plotters
 - Laserwriters
- Report generation – make the VI panel look like a report

Date	Time	Temp	Avg	Comment
12/9/90	9:19:27 PM	79.59	79.59	Normal
12/9/90	9:19:27 PM	79.59	79.59	Normal
12/9/90	9:19:28 PM	80.57	79.92	Normal
12/9/90	9:19:29 PM	82.52	80.89	Normal
12/9/90	9:19:30 PM	83.96	82.56	Normal
12/9/90	9:19:31 PM	84.47	83.64	Normal
12/9/90	9:19:33 PM	83.96	84.15	Normal
12/9/90	9:19:35 PM	83.96	84.15	Normal
12/9/90	9:19:34 PM	83.5	83.82	Normal
12/9/90	9:19:35 PM	83.5	83.84	Normal
12/9/90	9:19:36 PM	84.96	83.98	Normal
12/9/90	9:19:37 PM	85.94		
12/9/90	9:19:38 PM	86.43		
12/9/90	9:19:39 PM	86.91		
12/9/90	9:19:40 PM	86.91		
12/9/90	9:19:41 PM	87.4		
12/9/90	9:19:42 PM	86.91		
12/9/90	9:19:43 PM	86.43		
12/9/90	9:19:44 PM	85.94		
12/9/90	9:19:45 PM	85.46		



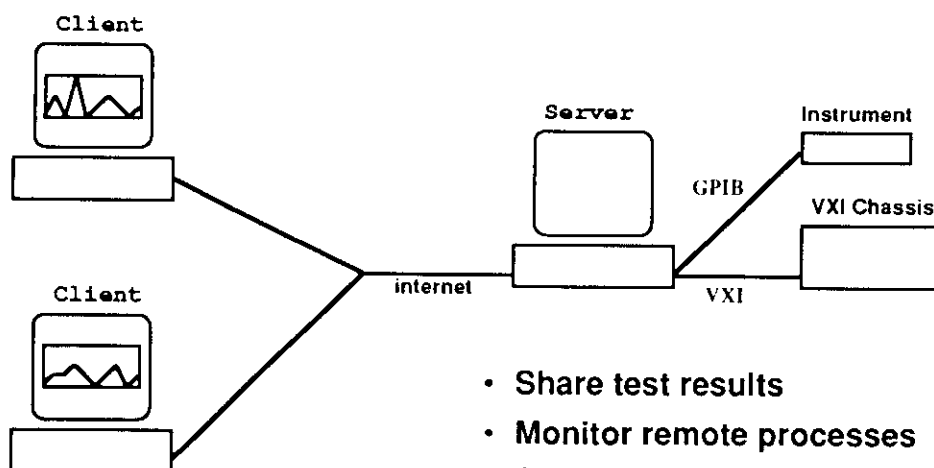
Slide 18 b)

Dynamic Data Exchange (DDE)



Slide 19 a)

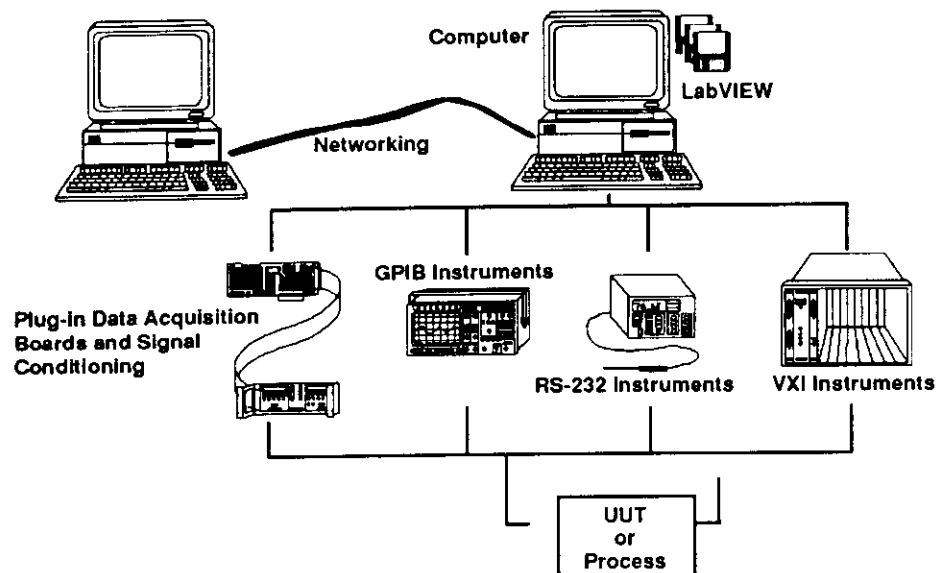
Networking in T&M



- Share test results
- Monitor remote processes
- Standard on Sun workstations
- LabVIEW supports TCP/IP

Slide 19 b)

Integrating Your Instrumentation



Slide 20 a)

Slide 20 b)