A LabVIEW™ Program for the Particle Analysis by Laser Mass Spectroscopy Instrument

by
David Thomson
Research Scientist - Systems Integrator
NOAA Aeronomy Laboratory - Original Code Consulting
and
Richard Winkler
Electronics Engineer
NOAA Aeronomy Laboratory

Products Used:

LabVIEW™
PC-TIO-10
PC-Servo-4A

The Challenge:

To develop software for the autonomous control and data acquisition functions of the Particle Analysis by Laser Mass Spectroscopy (PALMS) Instrument. This software must interface with dozens of commercial and custom devices, and must execute numerous asynchronous tasks in parallel.

The Solution:

Several features of the LabVIEW graphical programming language lent themselves to an efficient solution. The "open" nature of LabVIEW allowed us to easily create drivers for numerous commercial and custom instruments, as well as to effortlessly communicate with several NI hardware products. The inherent parallelism of LabVIEW was utilized on several levels to allow many asynchronous tasks to operate at the same time. Finally, all of this was accomplished in a fraction of the time that would have been required to create a program of this complexity with a more traditional language.

Introduction:

PALMS (Particle Analysis by Laser Mass Spectroscopy) is a unique, state-of-the-art instrument being developed by Dr. Daniel Murphy at the National Oceanic and Atmospheric Administration's Aeronomy Laboratory for determining in real-time the chemical composition of individual atmospheric particles. Although a prototype instrument has been operating for several years in the laboratory and in ground-level field campaigns, the current PALMS instrument is designed to operate in the stratosphere on the WB-57 high-altitude research aircraft. Since the WB-57 is a two-seater aircraft with numerous scientific instruments on board, each instrument must be nearly autonomous. Furthermore, due to the expense of flying the aircraft and the limited opportunities for flights, the software for the PALMS instrument must be very robust and include sophisticated error handling and recovery techniques. The task of writing a control and data acquisition program for the PALMS instrument was further complicated by the complex nature of PALMS, which includes over 30 pieces of hardware that must be under computer control. In addition, there are over a dozen states in which the PALMS instrument can be placed, and over half a dozen different tasks that must execute in parallel at various rates (some constant and some random) in order to optimize the operation of the instrument.

The PALMS Instrument:

The details of the design of the PALMS instrument have been described elsewhere in the literature[1,2], so only a brief description is presented here. Ambient air is sampled through an inlet in the nose of the WB-57, which can be seen in Figure 1. A capillary placed in the center of the flow of this inlet brings a small fraction of the particle-containing air into the PALMS vacuum system through a series of differentially-pumped orifices. As shown in Figure 2, particles within the vacuum system pass through a continuous laser beam and scatter light into a photo-multiplier tube. This scatter signal is digitized to yield an approximate measure of the particle diameter and also serves as a

trigger for the second laser. This excimer laser delivers several milli-Joules of ultraviolet light, which vaporizes and ionizes the particle. The resulting ions are accelerated down a time-of-flight mass spectrometer and are detected by

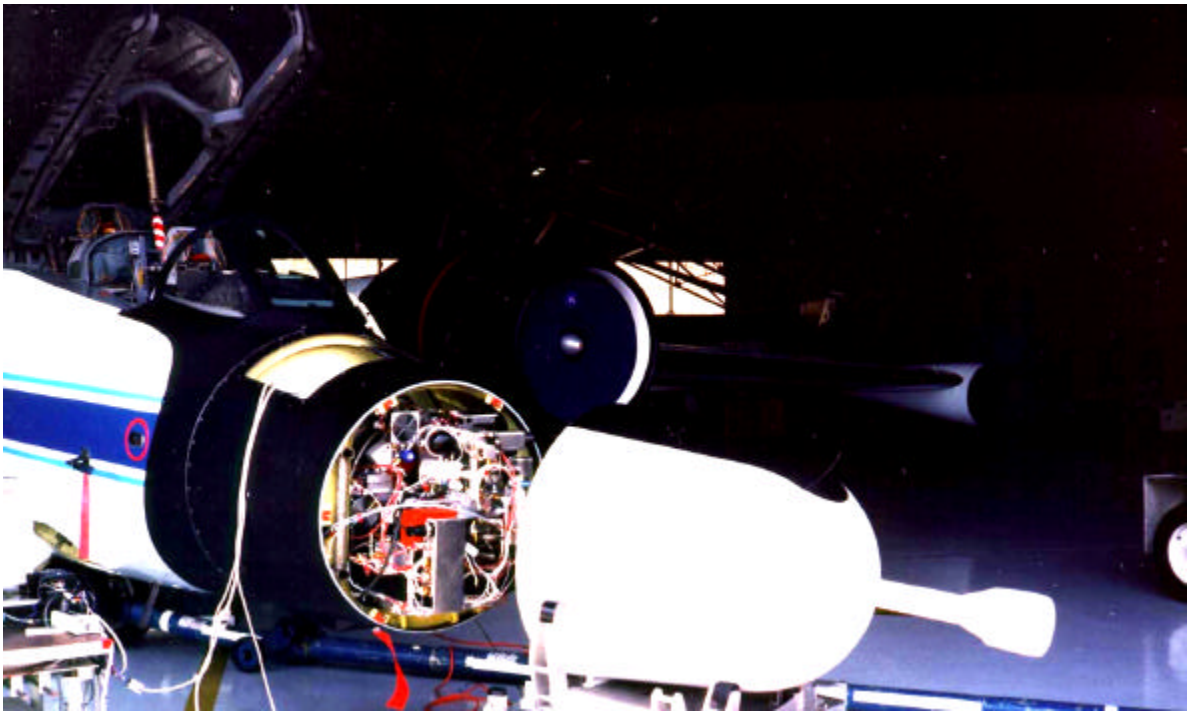

Figure 1. The PALMS instrument loaded into the nose of the WB-57 aircraft.

a micro-channel-plate detector. The entire mass spectrum for each particle is digitized and recorded. Analysis of these spectra yields information on the chemical composition of individual particles in the stratosphere and upper troposphere.

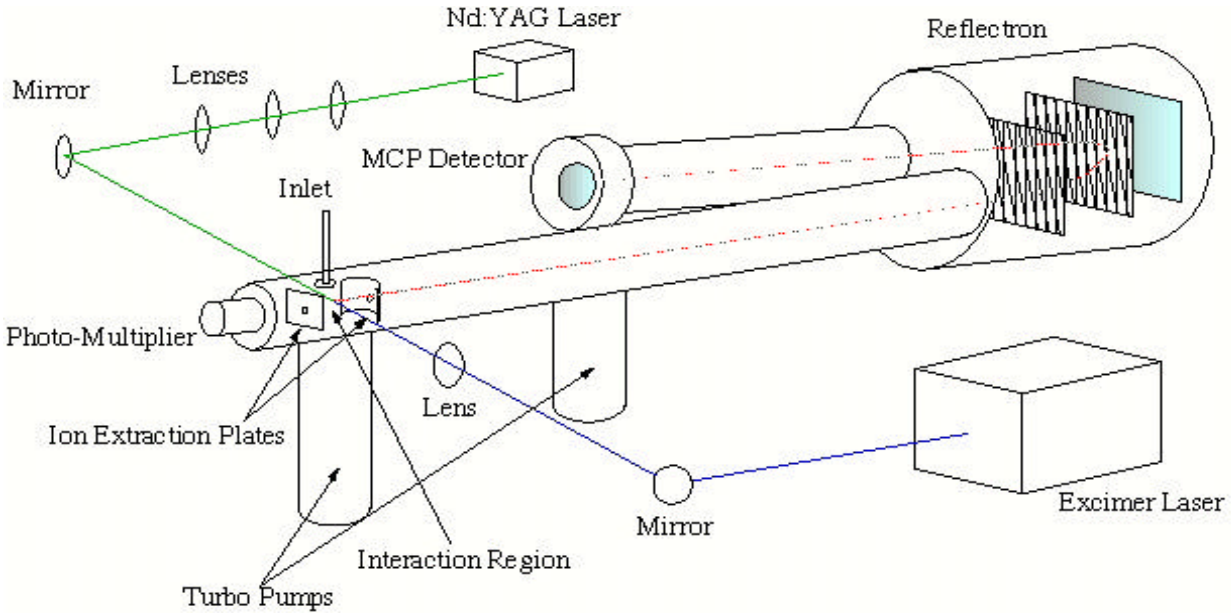

Figure 2. Schematic of the PALMS instrument.

Program Architecture:

In order for the PALMS instrument to operate efficiently, the following tasks must execute asynchronously at the intervals indicated in the Table 1. To accomplish this, a software architecture was developed that uses multiple top-level LabVIEW VI's running simultaneously. A main Control VI is launched when the computer boots. This VI reads a configuration file that describes the hardware in use, the sequence of states in which the instrument should be placed, and a list of additional tasks that should be launched. The additional tasks are loaded dynamically, so that tasks which are not needed (e.g. Operator Interaction during autonomous operation in flight) do not use up resources and degrade performance.

Task	Interval	Priority	Duties
Instrument Control	1-2 seconds	2	Determine and implement which state the instrument is in depending on programmed execution sequence, current alarm values, and limited co-pilot input.
Housekeeping Data	1-2 seconds	2	Acquire and log temperatures, voltages, pressures, etc.
Particle Data	Random (50 ms min.)	1	Acquire mass-spectral data of individual particles.
Laser Alignment	5 – 30 minutes	1	Measure and correct laser beam positions.
File Writing	As needed (20 ms min.)	0	Write data from other tasks to disk.
Operator Interaction	2 seconds	0	Allow manual control and monitoring on the ground.
Data Server	2 seconds	0	Distribute data to other instruments or observers.

Table 1. List of the asynchronous tasks run by the PALMS LabVIEW software program.

Although the various tasks operate asynchronously, they are very interdependent and must communicate with each other. Most of the communication is handled by passing strings of flattened data through FIFO message buffers. These queues are generally safer to use than global variables, since access to the data is more controlled. Also, by buffering the messages in queues, a slower task can eventually catch up on a backlog of messages being sent by a faster task, rather than acting only on the most recent message and missing the others. Global variables are used when appropriate, however, for handshaking and cases of data transfer where only the most current value of the data is of interest.

The priorities assigned to the tasks are critical to the overall performance of the system. The Instrument Control and Housekeeping tasks have the highest priorities, since Instrument Control is responsible for maintaining the hardware in a safe state, and the Housekeeping data is used to determine alarm values. The Particle Data and Laser Alignment tasks are mutually exclusive in their operation, and they share the next priority level. The File Writing, Operator Interaction, and Data Server tasks have the lowest priority. It is obvious that the last two of these have a low priority in order to keep human interaction from adversely affecting the data acquisition rate. The reason for a lower priority for the File Writing task may be less obvious, but is also related to achieving maximum data throughput. Writing data to disk is separated out as a distinct task since accessing the hard disk is a relatively time-consuming operation. Under the program architecture described, the Particle Data task can acquire a burst of data at its maximum rate and only have to store that data in memory by putting it in the File Writing task's message buffer. When the data rate has slowed to the point that the lower-priority File Writing task is finally given a time-slice, a portion of the accumulated data can be saved to disk.

This type of burst operation is an important feature of the PALMS instrument, since particles arrive in the instrument at random times. There are certain experiments where the ability to acquire a short burst of particle data is extremely important, as for instance when the WB-57 is attempting to sample an exhaust plume from a rocket or another aircraft. There are, however, several additional aspects of the PALMS software that allow this type of operation to work successfully. First, when using VI's of different priorities, it is important that all loops in the VI's include wait statements, so that a high priority VI cannot completely lock out lower priority VI's. Second, an additional safety mechanism was included in the Particle Data and File Writing tasks to handle the possibility of very large bursts of data. One part of this safety mechanism was to limit the maximum size of the File Writing task's message buffer, so that large bursts of data could not lead to an out-of-memory error. Another part was to incorporate a conditional wait within the Particle Data task, which only executes when more than a certain number of mass spectra have accumulated in the File Writing task's message buffer. This wait releases the higher priority Particle Data task and allows the lower priority File Writing task to catch up with some of its backlog.

In addition to having multiple top-level VI's running as parallel tasks, another level of parallelism was utilized within the main control task. Within this VI, three independent parallel loops are used. The first loop controls which state the instrument is in. Since initializing a given state can take several minutes, a second loop is used to continuously monitor the alarm values, co-pilot switches, programmed execution sequence, and possible manual control commands to determine whether the current state should be interrupted and another state initialized. A third loop continuously resets a watchdog timer, so that the entire system can be rebooted if the computer should lock up.

Instrument Drivers:

A large amount of the effort involved in writing the PALMS software went into writing instrument drivers for the numerous pieces of hardware required. Although some of the commercial hardware came with LabVIEW instrument drivers, many pieces did not, and much of the hardware was custom designed. The hardware that needed to be controlled included 104 bits of digital I/O, 96 A/D channels, 2 500 MSamples/sec digitizers, 7 serial ports (RS232, RS485, and SPI), 2 micro-positioners, 2 DC motors, 8 high voltage power supplies, 6 vacuum pumps, 2 lasers, and 3 laser power meters.

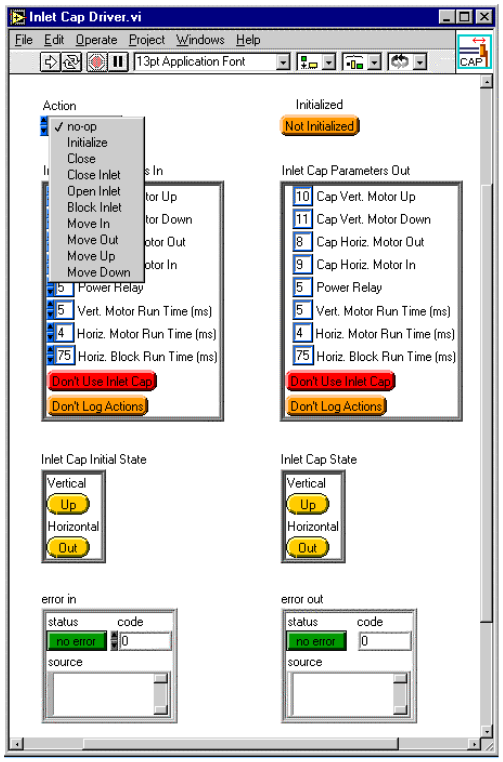

Figure 3. Front panel of the Inlet Cap Driver.vi.

Since many of the hardware devices need to be accessed from more than one of the independent tasks, the program architecture discussed above necessitated the development of a new device driver model. Not only could simultaneous calls to a single device from more than one task result in communication errors, but also a mechanism was needed for supplying each of the tasks with the configuration information necessary for accessing the hardware. Our solution was to create a high-level driver VI for each device. This driver contains code for accessing all the functionality of the device and maintains in local memory the configuration information for the device. By making all calls to the device through this single, non-re-entrant VI, multiple simultaneous calls are automatically serialized by the LabVIEW execution system.

As an example of this high-level driver model, Figure 3 shows the front panel of the Inlet Cap Driver.vi. This driver controls two DC motors that are used to seal the inlet to the vacuum system, as shown schematically in Figure 4. Although it is a very simple driver, it demonstrates the features of the high-level driver.

As seen in the pull-down menu in Figure 3, the Action control contains options for each function that the driver can perform, such as closing, opening, or blocking the inlet capillary, or moving either motor one direction. In addition, the first three actions are common to any such high-level driver. Of these first three actions, Initialize is used to send the configuration parameters to the driver, perform any necessary hardware initialization, and set the Initialized indicator to True. The initialization frame, shown in Figure 5, is the only place in which the Inlet Cap Parameters In and Inlet Cap Initial State controls are read: all other actions ignore these controls. The Close action is used to turn the hardware off and set the Initialized indicator to False. The no-op action is used to query the state of the driver and hardware without affecting the device in any way.

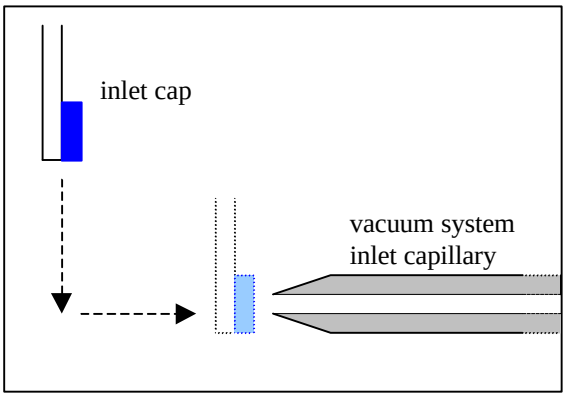

Figure 4. Schematic diagram of the functionality of the inlet cap system.

As shown in Figure 5, the Initialize action stores the configuration parameters in an uninitialized shift register.

This allows for the high-level driver to be initialized once by the main control task, then called by any other task without each task needing to know the configuration parameters. If another task attempts to call the driver before the control task has initialized it, the Initialized indicator is returned as False, and an error is returned in the error out cluster.

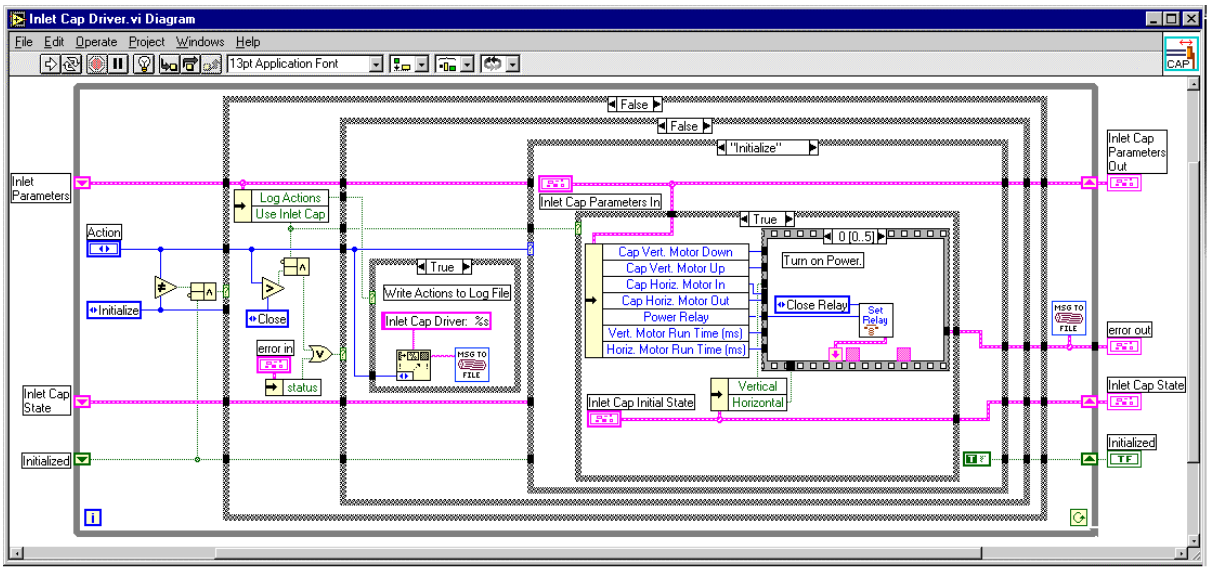

Figure 5. The block diagram of the Inlet Cap Driver.vi showing the Initialize action frame.

Other features of the high-level driver model are the manner in which errors are handled and actions are logged and the ability to disable the hardware via the input parameters. Since PALMS is a complex and autonomous instrument, it is important, especially in the early stages of its use, for the software to generate a log file describing what actions were being carried out at what time. Writing to this log file can be turned off for each high-level driver by setting the appropriate input parameter to Don't Log Actions. The log file is also used to record error messages. For this type of autonomous instrument, it is important that dialog windows never appear, since in normal operation there will be no user available to respond to the dialog and the program execution will halt. As an alternative to the normal dialogs, logging errors at the end of each high-level driver provides a consistent way of recording them. Additionally, error handling and recovery is implemented within each high-level driver or in the calling program as appropriate for each device.

By setting the appropriate input parameter to Don't Use Inlet Cap, the high level driver can be configured to ignore the inlet cap hardware. When used in this manner, any action requested of the driver which normally accesses the hardware is ignored, but no errors are reported. This mode of operation allows the program to run with various pieces of hardware disconnected or otherwise non-operational so that portions of the instrument can be tested independently.

One disadvantage of this high-level driver model arises when the device can supply numerous responses of different data types when queried in different ways. The current solution in this case, which is not demonstrated by the Inlet Cap Driver.vi, is to create an indicator for each of the types of responses. Depending on the complexity of the device, this can lead to cluttered front panels and connector panes. Furthermore, the programmer must keep in mind that each indicator is valid only for certain actions, and that invalid indicators will contain default or irrelevant data.

Conclusions:

LabVIEW proved to be an ideal programming language for the development of the PALMS instrument data acquisition and control program. The ease with which parallel tasks are implemented within LabVIEW was crucial to writing an efficient program in the time available. By using several independent tasks, various priorities could be set for the different asynchronous operations, allowing them to be optimized for data throughput and safe instrument operation. In addition, parallel loops were used within a task to keep related but asynchronous operations from holding each other up.

A complication that arose from dividing the PALMS software into numerous tasks was that several tasks could potentially attempt to access the same hardware at the same time. To avoid this problem and to simplify the manner in which the hardware configuration information was distributed to each task, a new high-level driver model was developed. By including all the functionality of the device in one non-re-entrant VI, simultaneous calls to the same device are automatically serialized by the LabVIEW execution system. This high-level driver model also incorporates some general features, such as action and error logging, which are useful for this type of complex, autonomous instrument.

1) D.S. Thomson, A. M. Middlebrook, and D.M. Murphy (1997) Thresholds for Laser-Induced Ion Formation from Aerosols in a Vacuum Using Ultraviolet and Vacuum-Ultraviolet Laser Wavelengths, Aerosol Sci. Technol., **26**, 544-559.
2) D.M. Murphy and D.S. Thomson (1995) Laser Ionization Mass Spectroscopy of Single Aerosol Particles, Aerosol Sci. Technol. **22**, 237-249.