MICROSTAR LABORATORIES™
onboard intelligence since 1982

2265 116th Avenue N.E., Bellevue, WA 98004
Sales and Customer Support: (425) 453-2345
Finance and Administration: (425) 453-9489
Fax: (425) 453-3199
WWW: http://www.mstarlabs.com/

From the Microstar Laboratories web site

Implementing Automatic Controls Using Data Acquisition Processor Boards

Though Data Acquisition Processor (DAP) boards are primarily data acquisition devices, the onboard processing power that makes rapid service to data conversion and buffering devices possible can also be applied for rapid service to external control processes. There are certain limitations, but for systems that can stay within these limitations, Data Acquisition Processors provide a reliable platform for the implementation of control methods that would be very difficult or very expensive to deliver any other way. Such systems might implement advanced control methods, require special data analysis, or operate on a scale beyond the capabilities of ordinary control devices.

This white paper surveys the challenges and strategies for implementing automatic control systems on Data Acquisition Processors.

Contents

Applicability

Data Acquisition Processors provide device-level controls. They can operate in stand-alone workstation nodes, or as low-level remote processing nodes within a multi-level control hierarchy. The fact that they can capture a large number of measurements accurately means that they are useful as data server nodes in SCADA networks, with the host supporting the SCADA protocols and network connections. They can apply local process knowledge to implement a control policy downloaded from a higher level node, receive and respond to feedback, and intelligently analyze data so that higher level controls receive the best information.

Perhaps it is easier to say when Data Acquisition Processors are not appropriate. Applications with relatively low measurement rates, moderate accuracy requirements, one or very few measurement channels, and minimal processing needs will probably find alternative low-cost devices that serve the purpose well. Most feedback regulator controls have pre-packaged solutions available at a reasonable cost. Even though this means that the great majority of measurement and control applications do not need the high-end capabilities of Data Acquisition Processors, this still leaves many difficult measurement and control applications for which Data Acquisition Processors are well-suited but for which few alternatives are available.

In addition to basic requirements for signal capture, control applications must process those measurements, apply a control policy, and generate control responses, meeting real-time constraints while doing these things.

The simplicity of pre-packaged control solutions offers very little in the way of extensibility or flexibility. For those cases in which the most basic control solutions are not sufficient, the options are few.

1. For optimal performance, but with the highest cost, greatest risk, and least flexibility, hardware-software solutions can be designed from the "ground up." The difficulties of this are great enough to make this option *out of the question* in most cases, but the benefits can be great if it is possible to apply the results widely and amortize the development costs.

2. To accelerate the development cycle and reduce costs, it is often worth compromising slightly on performance and using modular hardware and software sub-systems rather than starting everything from scratch. Data acquisition components, interconnected systems, and processor modules are available. Finding components that can inter-operate, can meet performance specifications adequately, and are supported by sufficient

and affordable development tools can remain a challenge. Long-term maintenance is a concern.

3. Data Acquisition Processors are the next step in modularization. They provide data conversions, processing power, signal interconnects, and a straightforward development platform. The hardware is completely pre-configured, and so is the software necessary for device operation, but this doesn't leave you much hardware-level flexibility. If you can work within the architecture, it is relatively easy to program specialized control features using a high level programming language. Pre-existing software provides for routine configuration of timing rates, data channels, data routing, pre-processing, and so forth. When you have completed the software configuration and customized control software, the hardware is an off-the-shelf commercial component ready to install.

4. Or in the end, you can stay with simple control devices that are available, cope with their limitations, and bear the economic costs of operation far below optimal capacity.

The Role of Architecture

Data Acquisition Processors are designed for measurement processing, where *speed* means lots of channels and bulk data movement. Devices that best facilitate bulk data movement are not the ones that best facilitate responses to individual items. Consequently, there are limitations on the speed and regularity of process timing to be considered.

When processing captured data streams at very high rates, a delay of a millisecond causes backlogs of thousands of samples in hardware devices. Both conversion inaccuracy and loss of data are unacceptable. This suggests the following criterion for evaluating real-time performance: response times must be in fractions of a millisecond. In contrast, workstation systems must operate on a human time scale, and delays much longer than 50 milliseconds make system response seem sluggish. This establishes a comparable criterion for its real-time performance.

Delays for processing and moving data make the problems of converter device operation more difficult. Processing is interleaved, with some processing time applied to device operation, some to data management, some to data communication, some to higher-level processing. It takes time for data to propagate through these stages. As a general rule of thumb, you cannot depend on a Data Acquisition Processor to deliver a completed result in response to an individual sample in less than 100 microseconds (and you must verify by testing).

Contrast this to the architecture of a typical workstation host. It is optimized for moving bulk data to support graphical data presentations and multimedia. After operations are set up, data movement is fast, but chip sets for memory management, bus control, and peripheral control require setup times in preparation for the data movement. There is even a setup time for an operation as simple as reading the current system time from an external timer chip. Workstation systems typically host an array of diverse devices, and these devices compete for service. There is no assurance about how much time the devices will take, or how much time device driver code will take to operate them. Add to these difficulties the heavy computational loading of a typical workstation operating system, and perhaps a GUI environment specialized for data presentation, and in the end you can't depend on delivery of any data in less than about 10 milliseconds... Or it might be 100 milliseconds or longer depending on a variety of uncontrolled conditions.

Even if you replace the operating system with a fine-tuned embedded microkernel system specialized for real-time systems, disable all unused devices, eliminate user interaction, and devise a way to load your control software as an embedded application, even in this highly customized PC environment, you will still have a very difficult time getting responses for the *best case* in less than a millisecond or two, and even then you can't trust it. It simply isn't designed for this purpose.

The Role of Processing

On a Data Acquisition Processor, the hardware processes for measurement and the low-level software processes to capture and move that data are pre-programmed. Control algorithms will reside in higher-level processing organized as independent processing tasks, which are scheduled for execution by a system task scheduler. There isn't much overhead with any of these operations, but there is some overhead with each, and the effects on delay are additive.

Data movement mechanisms called *data pipes* will transfer data between tasks or to external devices. The schedulability of the tasks depends on availability of data.

Timing and Regularity

Basically, you do not need to worry about obtaining samples consistently and accurately on a Data Acquisition Processor. A hardware-based sampling clock controls the timing of sample capture when an input sampling configuration is started.

Sampled data systems depend on frequent and regular sample values. While the sampling is hardware controlled, the control processing is not directly tied to the hardware oscillator. Collected data will be moved out of hardware devices and into a memory buffer where they can be managed by the onboard processor. The pipe system will then make the data available to processing tasks. Control processing can then begin to respond to the new data.

Collecting data in hardware and managing it in blocks with a general purpose processor has big advantages for general data acquisition, which is of course what Data Acquisition Processors are primarily about. But data collection and buffering can be an undesirable level of "baggage handling" for the case of high-speed control applications. Collecting samples in memory means delays, which are undesirable in many ways. But for applications where these delays are small compared to the time scale of system operation, the ability to collect a block of data spanning many signal channels, and then responding block by block instead of sample by sample, delivers a large boost in efficiency.

Ideally, all previous processing is completed and tasks are idling at the time that a new sample becomes available. This sample is plucked immediately from the hardware buffering devices. Processing of that value is completed and responses are delivered before additional sample values have time to accumulate. There are delays, but these are bounded, and there are not many of them. If the interval between samples is longer than the maximum possible delay plus the processing time, responses for each sample are guaranteed to be delivered before the next sample is captured. Under these conditions, no supplementary synchronization mechanism is required. To respond to new data in minimum time, all that is necessary is to attempt to read new data, and that request will be completed as soon as possible after the data become available.

Sampling and Bandwidth

Before worrying about what the Data Acquisition Processor will do with data, it is important to determine the application needs first. The sampling rate is important.

For the case of digital systems, you will want to watch hold times closely. When update cycles are fast, you might want to deliberately hold digital output levels for an extra update cycle. Otherwise, delay variability might cause very late activation in the current cycle but very early deactivation in the next, leaving a very narrow pulse that an external system might miss. For input data, there is a similar hazard. If the digital level is not held for a full sampling interval or more, it is possible for sampling to miss a pulse completely.

For continuous signals, there is the fundamental *Nyquist Limit*. Approaching this limit, the fundamental one-sample delay inherent in digital systems starts to become as important as the real content of the signals. Even if all of the important frequencies are below the Nyquist bound, it is still possible for the aliasing phenomenon to track higher harmonic frequencies if they are present, producing misleading effects in the sample sequence. For digital to analog output conversions, the latching converters act as a zero-order hold, and operating them too slowly produces a very poor stepped approximation to the desired output signal. You can perform separate simulation studies to optimize timing rates, but generally for output signals that are reasonably smooth the sample rate should be at least 10 times larger than the highest relevant frequency in theresponse spectrum.

Pipelining Delays

There is always an inherent one-sample delay in a discrete control loop. Given that an output was applied at the current time, the control loop will not see the feedback results until the next sample is captured.

Additional delays include the conversion time, the time for processing to detect new samples and access the hardware devices, data transfer delays to place the sample into memory and attach this memory to the pipe system. These delays are fundamental limitations of the Data Acquisition Processors and the DAPL system. They occur before processing can begin.

Task Scheduling Delays

Existence of data for control processing does not in itself guarantee that the control processing begins execution immediately. Task scheduling introduces additional delays. In a high-performance control application, there should be a minimum number of other tasks interfering with the execution of the critical control sequence. System-level tasks for purposes of device control are always present and these you cannot control. There are application-specific delays associated with completion of the control processing, and you do have some control over these delays.

Some means for controlling scheduling delays are:

- Use the minimum number of tasks consistent with application requirements and acceptable software complexity.
- Deliver responses as soon as possible. If any processing can be deferred until after the result is delivered, do so.
- Optimize the execution path between the detection of new data and the delivery of the control result.
- Break up processing of long numerical sequences with `OPTION QUANTUM=200.`

- Where you can, use lower priority processing for tasks that might interfere with critical time response.

Real-time constraints are less about raw speed than about what delays might happen, how often they happen, and how corrections are applied when they happen. Applications will usually fall into one of these categories:

1. very short delays never matter,
2. short delays do not matter as long as they occur only occasionally and recovery is fast after each,
3. delays beyond the fundamental one-sample delay are not tolerated.

When more than one sample has time to collect before control processing is scheduled, your control application will receive more than one new input value. Your processing can spin through the backlog of values quickly to "catch up" to current time, losing no information and bringing state up to current, but it cannot go back to generate missed outputs. For applications where these delays are not long enough or frequent enough to matter, this is a natural means of recovery and you won't need to take any other corrective action.

If you are willing to tolerate the delays, you might be able to operate at higher sampling rates, but a consistent backlog of data will result in a consistent output delay. Sometimes this backlog merely cancels out any speed increases you might have thought you were getting because of the higher sampling rate. At other times, the increased processing overhead will mean that there is a net increase in average delay time, and this could increase the phase lag enough to compromise the stability margin. You might have to operate the loop at a lower gain setting to compensate, with reduced disturbance rejection at the lowered gain.

Time Delay Benchmarks

There is a limited amount that you can do to avoid delays, but the DAPL system can provide you with information about what those delays are. You will need to simulate system timing with executable processing tasks. These do not need to be the actual processing tasks, particularly not in early project stages while feasibility is being evaluated. This is actually a rather important point: *you do not need to fully implement your system to study critical timing.*

- To simulate processor loading, you will need a custom processing command that wastes a known amount of processor time for each sample that it reads. You can get this using a loop that writes to a hardware device repeatedly, for example, using a `dac_out` function. (Putting a hardware function in the loop prevents the compiler from "optimizing away" your processing code and eliminating the processor loading that you want.) Suppose for example that you observe that, on your DAP model, the loop can execute 2.5 million times per second. Then invoking the loop for 2500 cycles will produce a simulated 1 millisecond processing activity.

- To simulate system processing delays, determine how many processing tasks you will have, and produce a network of tasks that move data through the pipes in much the same configuration that your application will use. Lay out your application data flow sequence, and substitute arbitrary processing for actual processing that is not yet available. For tasks that are primarily data movement, use `COPY` commands. For tasks that are primarily data selection, use `SKIP` commands. For tasks that are primarily numerical processing, substitute `FIRFILTER` or `FFT` commands.
- To simulate delays from transfers to the host, you can use a `COPY(IPIPE0,$Binout)` processing command, and configure DAPstudio software to receive and discard the data.
- Try to use a realistic number of samples per second in each data transfer.

Once you have a processing configuration to test, you can activate the built-in task statistics monitor. Documentation on this is provided in the DAPL Manual. In DAPstudio, you can right-click the Processing configuration tab, and, in the *Options* dialog, activate the *Sequencing* option. This will give you a new *Sequencing* sub-tab under the *Processing* tab. In the sequencing editor dialog, you can add the following lines to your processing sequence, after the `start` command.

```
statistics  on
pause 10000
statistics
statistics  off
```

This will monitor task timing for a period of 10 seconds, displaying the timing statistics for all tasks. The most important statistics:

1. Compare the amount of time used by the task to the total run time. This will verify assumptions about the amount of processor loading in each task.
2. Check the worst-case latency. All tasks in one processing loop will have the same latency, so the worst case for one is the worst case for all. If this time interval is less than the sampling interval, there will never be a data backlog, and the most data received at any time will be one new sample.

3. If the worst-case time is less than two sample intervals, and the average processing time is much less than one time interval, there will be infrequent backlogs of no more than 1 extra sample.

Multitasking Control

Some control activities are critical for delivering a response as quickly as possible, while other activities perform the analytical work to prepare for this. It is possible to take advantage of multitasking features of DAPL to separate these activities so that there is minimum interference with time-critical processing.

If you can use tasking priorities, you can place the time-critical processing at a high priority level and leave the preparation and analysis processing at lower priority. When you do this, the lower-level tasks have negligible effects on process timing, yet they will use any available time for completing their processing in the background. The disadvantage of this scheme is that the tasks are completely independent, so sharing state information can be complicated. You might need to keep some of the preparatory state-management code in the high-priority task, but perform those computations after time-critical processing is finished.

Representing State

Almost without exception, controllers maintain some kind of state information. Even model-free control schemes such as PID maintain derivative and integral estimator states. Each control action depends on a combination of new input information and state information.

It is possible to make better control decisions when those decisions are based on knowledge of how the particular system responds. State information can be used to adjust for characteristics of the input sequence, the characteristics of output actuator devices, and estimate unmeasured internal variables.

Logic Control and Discrete Events

Programmable Logic Controller devices exemplify discrete control strategies with limited state information. For example, if switch *A* is closed, take action *B*, otherwise make no change.

Sequence control is not the specialty of Data Acquisition Processors. However, the ability to apply actions in a flexible manner, that is not locked to a fixed updating cycle, can be an advantage. Based on the current discrete state, the appropriate processing code is selected and appropriate state variables will be updated. Discrete states will typically be represented as a combination of state data with associated processing sequences.

In the simplest case, the process coding maintains a state enumerator code and selects processing according to the current value.

```
...
pipe_value_get(pPipe, &new_input);
switch  (state_enum)
{
  case STATE_1: ...
    break;
  case STATE_2:
    new_out = apply_control( new_input,  &state );
    send_response( port, new_out );
    task_switch();
    state_enum = next_state( new_input,  &state );
    break;
  case STATE_3: ...
    break;
  ...
}
```

In the above sequence, the new input value is used to generate the output response as quickly as possible. After that, the task voluntarily releases the process control so that if there are any other high-priority control tasks that need to respond quickly, they will have an opportunity to do so. At the next opportunity to execute, the task updates its state information and prepares to process the next event.

Continuous State and Updating

Internal dynamics of many complex systems are modelled by differential equation systems in continuous variables. Mechanical systems are typically well approximated by multi-variable linear differential equations. Other kinds of systems, such as biological and chemical processes, usually require nonlinear differential equation systems. When the sampling rate is sufficiently high, a discrete-time approximation to the continuous equations performs well. A separate numerical analysis can show that the equation system works correctly with the sample interval chosen.

Critical-time processing will collect the most current measurements of input and state variables and evaluate the output control rule equations using these variables. Follow-up or lower-priority processing will use the current input and output information and the differential equation system to determinethe next control state.

Complex State and Subtasking

Complex control problems often involve "operating modes" with different control objectives: seek a resting position, track a changing reference level, flush reagent, vent steam for fast overpressure reduction, etc. These problems involve a combination of discrete processing and also continuous state updates. Some state variables will be maintained at all times, and others become relevant only for certain operating modes. Keeping track of this complexity will typically force the sequencing logic to be defined in one processing command. A dispatcher section is activated for each input sample, and it selects appropriate processing based on current state information. The more complex the control scheme, the more difficult it will be to partition into simpler independent processing commands.

Systems with a complex state are often associated with complex data representations. As much as you can, you must avoid navigating through complex data structures in the critical response sequences. For example, suppose that the arrival of the next cyclic scheduling event will mean that it is time to check the process limits on a level sensor. What are the limit levels for this sensor? To determine this, you need to use the device identification to locate the setting in the limits table. How do you tell which conversion gains to use for this sensor's readings? You need to index the device's calibration settings in the calibration table. If a limit is reached, what output signal receives notification? The response option is located in the response options table, and from the response option, you can locate the entry point for the limit-handling process. There is nothing wrong with any of this, but indexing through the data structures can result in swapping data in and out of the processor cache, which in itself can mean a factor of 10 increase in processing time.

To avoid spending this data management time during critical response intervals, you can borrow a strategy from database systems. In database systems, going through all of the data tables in response to each query can result in very poor response time. If it is known that a type of query will occur, the access to the data can be pre-packaged into a *view*, something like an artist's palette, with just the bits that are currently needed. More specifically for control systems, a temporary data record can be prepared with just the information needed for the service of the next event. After the response is issued, the consistency between the temporary structure and the main data structures can be restored, and preparations made for the next update.

Measurement and Observation

Data Acquisition Processors are natural tools for acquiring data and applying pre-processing operations necessary for the best measurement quality. This processing can include statistical reduction, such as averaging or filtering, or advanced estimation techniques for deriving information that is not directly measured.

Oversampling, Conditioning, and Calibration

When using filtering for noise reduction and anti-aliasing, it is a common practice to capture samples at a higher rate, then reduce the rate by decimation after filtering is applied. Most, or perhaps all, of the high frequency signal sources that might corrupt the control loop data are represented accurately at the high sampling rate, and filtering of the digital signal reduces or removes them.

There is little additional timing uncertainty introduced to Data Acquisition Processor processing when the filtering is used. The time to access one value or to access a data block is not much different. There is an additional task to perform the filtering, however, and the scheduling of this task will cause a predictable delay of about 10 microseconds, plus the time for a pipe operation, in addition to increased processing time.

Calibration is an attempt to correct for persistent differences between expected device performance and actual measurements, based on certain reference conditions such as constant known input. Calibration is often applied to "linearize" signals from sensors, adjust for gain variations, and cancel constant offsets. Some calibration tests can be programmed and applied automatically as part of the lower-priority processing and filtering.

State Observers

It is sometimes very difficult or very expensive to completely measure all potentially relevant system variables. Sometimes the relationship is relatively simple: for example, if you have the complete position history of an actuator device, you can derive a quite good velocity history as well. Even simple controllers such as PID will estimate the derivative of the observed signal as part of its strategy.

When the controlled system is observable, and a good dynamic model of the plant is known, in theory it is possible to reconstruct the complete state given a fixed number of values from the input/output history. In practice, noise will disrupt this process, so the estimates of unobserved state variables are not perfect. Still, a good estimate can be useful, and much less expensive than complex instrumentation.

Output and Actuators

Actuator devices sometimes have their own important characteristics and dynamic states. In some instances it is useful to compensate for these characteristics using an independent level of control processing.

Actuators can be treated almost as separate control problems. The time-critical part generates the compensated drive signal after receiving the command level from the control algorithm. At a lower priority, analysis of state and nonlinearities determines how to respond the next time.

Control Strategies

There are many different control strategies. We can list a few examples here and relate them to the implementation strategies previously described.

Model-Free

Model-free controllers use various linear or nonlinear mappings from input values and state values without explicitly representing a system model. The most common is PID control. Some generalized mapping controllers such as neural networks are in this class.

Unless there are unusual requirements for very high rate updates, diligent monitoring, large numbers of channels, etc., Data Acquisition Processors are usually not the first choice for the simple and conventional model-free approaches such as PID. It is not unusual, however, that slight variations are required, and, while these can be difficult to implement on other platforms, they are almost trivial with DAPL processing commands.

Model-Based

Model-based controllers have control laws with special structures matched to the properties of the system at design time. On the theory that there is a better chance of controlling a known system than an unknown one, a custom-matched controller design should work better than a "generic black box" solution. The cost is primarily in the engineering analysis. The resulting control law, in whatever its special form, is typically not much more difficult to apply than PID control.

Model-Reference

Certain strategies for complex systems incorporate a simulation (sometimes reduced order) that explicitly represents the system, and then bases control action on a comparison between the simulated and actual system response to observed inputs. The simulation can also be used to project future effects of present-time control decisions. There is a close relationship to state observer methods. The time-critical control rule for using the state information is usually very simple in comparison to the background evaluation of the model. Obtaining a useful system model is not an easy problem in general.

Adaptive

For systems with known form but poorly known parameter values, or systems that have properties that change over time, numerical techniques for estimating parameters dynamically are well known. These computations can require a lot of processing over an extended time, but with no urgency and no real-time deadlines. Lower-priority processing tasks in the DAPL system work well for this.

Adaptive controllers are a complement to model-reference and observer-based controllers. Those controllers presume that a discrepancy between the observed output and the model is due to a disturbed state that must be corrected. Adaptive controllers, on the other hand, presume that discrepancies in the observed output are due to deficiencies in the model, and that better model parameters will produce improved performance in the future. Model-free controllers can use a direct-adaptive approach, adjusting gains and other control parameters directly in an attempt to improve control performance without reference to an explicit model, but the effects of controller parameters can be difficult or impossible to distinguish from unknown system parameters in a closed loop configuration, so proving convergence can be difficult.

Noisy Systems

For systems in which signal measurements have a large amount of noise, the control strategy can take variability of the measurements and resultant variability in state estimates into account. Optimal controllers based on the classic Kalman Filter and the Extended Kalman Filter are of this class. These designs iteratively solve a separate, explicit model of random noise processes and use that to guide state estimate correction. Noise increases the difficulty of obtaining suitable models and limits applicability.

Contrary to the popular myth, these controllers typically are *not* adaptive. The state is adjusted, but the model parameters are not. Kalman filtering techniques can be used in combination with adaptive methods, but that is relatively uncommon.

Both the state update and the state correction update require matrix mathematics. A considerable amount of processing power is required. The floating point hardware support provided by a Data Acquisition Processor main processor chip is very helpful for this.

Feedforward

Given a desired system output trajectory, pre-processing of a command signal can be very effective in driving a system along that trajectory with minimum excitation of undesirable oscillatory modes. DAPL processing tasks are particularly good at this kind of processing.

A weakness of feedforward controls is that they cannot see the effects of disturbances, so they typically must be used in combination with stabilization and feedback correction controls. That combination can be more effective than feedback controls alone.

Nonlinear

Every system is nonlinear to some degree, and many optimal controllers are nonlinear. Nonlinear control strategies can be used to compensate for system nonlinearities. Even when the system is linear, a nonlinear control strategy can be applied to improve performance.

As an example of nonlinearity in a system, thermal leakage flow in a refrigeration system is one directional and varying, from a warmer ambient environment into the refrigerated chamber. To make the refrigerated temperature colder takes effort to overcome the thermal leakage, whereas thermal leakage can raise the temperature by itself if no action is taken. A controller that responds unequally to colder and warmer temperature deviations can compensate for the imbalance, improving regulation.

An example of nonlinear control of linear systems is the "California Driver" strategy. To reach a destination as quickly as possible, stomp on the fuel pedal until the last possible instant, at which time you stomp down on the brake and bring the vehicle to a halt exactly at the destination. As harrowing as the strategy might be, it works. You will spend the least time accelerating and also the least time decelerating, so you arrive as quickly as possible. In contrast, a PID control policy would gradually remove fuel or braking action as the distance to the destination decreased, allowing the vehicle to drift gradually to the final destination, but taking more time. A mixed strategy can be applied, but the mixing process is itself a kind of nonlinearity.

Fuzzy Controls

Control problems that have multiple operating regimes can sometimes use the fuzzy control formalism to apply relatively simple control strategies in combination, with the relative "membership" of fuzzy decision variables determining the mix of actions to apply.

While very general, fuzzy systems require additional mappings to convert from measurement to internal variables, and from internal variables to outputs. Fuzzy rules have the property of always applying, but to a lesser or greater degree, so every possible action is considered all of the time, possibly contributing zero to the result. Consequently, the extreme generality of fuzzy control comes at a significant cost.

If you have multiple fuzzy control commands, combining the commands in a single downloadable module allows them to share the fuzzy inference engine code, but not the rule sets or state. Each task will instantiate its own independent rule base and evaluation state.

Tuning and Monitoring

Most controls are self-contained units. Most settings must be applied physically at the device location. With Data Acquisition Processors, evaluating and adjusting settings is a software function, so the Data Acquisition Processor can assist with the measurement, application, and configuration management. The actual monitoring can be done at the local station or controlled from a remote location. While tuning and monitoring are not central to the real-time control problem, they can be central to the problem of controlling operating costs.

Passive Observation and Logging

Is the loop alive? Is the loop processing normally? Is the loop processing at a sub-optimal level?

Along with control responses in real time, Data Acquisition Processors can collect, reduce, summarize, and deliver summary data to the host system or a remote host to analyze and log. This does not take much computation, and Data Acquisition Processors can organize the data collection without much effect on the time-critical processing. Apply one DAP to multiple channels, with both control and process monitoring, and suddenly this begins to look like a significant cost improvement.

The software support for Data Acquisition Processor operation transparently includes networking support. Provided that you have communications channels available, a local processing node can transfer observed data to another location for recording, display, or interactive analysis.

Opportunistic Self-Testing

When using model-reference processing, unexpectedly large differences between what the model predicts and what the system actually does can indicate that something in the system has changed and needs attention.

There is not much that can be learned while a loop sits at a regulated level with no disturbance, but response to disturbances can provide a lot of good information. Processing can be configured to identify disturbance events and apply special analysis or actions when these occur.

Self Testing with Active Injection

While a controller holds a loop at a regulated level, it is possible for conditions in the system to change so that the control settings are no longer close to optimal, and stability may be compromised. Instead of waiting for disturbances to occur, and discovering too late that the system is in trouble, it is possible to inject a small disturbance deliberately. The injection of the disturbance signal into the control output is part of the time-critical processing, and must be integrated with the control loop update. Preparation of the injection signal can be coordinated by a lower priority process. Tests can be initiated periodically, under local control or at the request of higher control levels.

Where the system is sufficiently linear, an injection experiment can be a small disturbance of short duration, riding on top of the normal command level. Superimposition principles can be applied to isolate the disturbance's dynamic response from the static operating level, yielding information about open and closed loop characteristics. It is not necessary to remove the loop from active service for the purposes of this testing, and the testing activity can be controlled remotely.

Fail-Safe Monitoring and Recovery

If devices have failed, the process most likely needs an orderly shutdown. This kind of processing is typically enforced by a watchdog timer. Watchdog timers require very little processing, but when they do activate, their actions should be delivered at a high priority so that they are not unduly delayed by routine data processing.

To use watchdog monitoring, an independent task can watch a data source. The task requests a `task_pause` interval, using the DAPL system timer. While inactive, the task causes no scheduling delays. When the task next executes, it expects a non-zero code. If it sees one, all is well. It discards or clears the value, then activates the next `task_pause` interval. But if it fails to see the expected value, there must be a fault because otherwise a sending task would have provided a timely update.

Fault indications will sometimes require signal channels separate from ordinary data logging. They can be hardware devices such as digital outputs, but they can also be messages sent on a communication channel. In Data Acquisition Processor systems, these communication channels are "virtual" in the sense that the actual channels pass through the same physical network medium as any other data transfers.

Special Applications

This section mentions a few examples of applications with characteristics that make them particularly challenging, yet not terribly difficult for a Data Acquisition Processor.

Distributed Process Control

For large scale systems, it is important to subdivide the plant control objective into subproblems that can be addressed individually at local processing nodes. At these local processing nodes, the host software can further distribute the controller commands to multiple Data Acquisition Processors, each controlling a cluster of control channels.

It is unusual to find an architecture that can scale up to manage many control channels almost without limit. DAP processing actually gets more efficient when supporting arrays of samples rather than just one, yet each channel can have an independently adjusted configuration.

Hydraulics

Advanced hydraulics controls such as shaker table actuators drive high forces and respond over a frequency band much too wide for most ordinary controllers. The hydraulics systems have multiple servo stages, with nonlinear seal friction effects and a tendency to "ring" at around 5 to 15 Hz. Having the speed of a DAP system for sampling intervals of 1 millisecond or shorter is important for controlling these systems to full bandwidth. DAPs have the processing power to implement effective nonlinear control methods with multi-variable feedback.

Spatially Distributed Processes

Processes that are physically distributed, such as a rolling mill, drying conveyor, or distillation column, need arrays of sensors, and they must operate arrays of control devices. Decisions made early in the process propagate through the system and directly affect downline decisions, while requirements of downline processing can feed back to influence the process decisions at the start of the line. A control strategy will involve timed and coordinated actions. Data acquisition processors can provide effective measurement and management of multiple-channel data, the ability to schedule delayed actions, and packaging of local and global processing levels as different tasks.

Conclusions

We have seen that Data Acquisition Processors offer a fixed architecture solution that, while originally intended for classic data acquisition, also has application for advanced control systems. The fixed architecture means that the hardware development problems are reduced to establishing and configuring good interconnections. The software problem is reduced primarily to implementation of the control strategy, with concerns about the distribution of critical and non-critical processing times. The disadvantage is that you must cope with some unavoidable timing delays that are always present, along with some additional delays that your control processing will introduce. The net delay depends on the amount of data traffic and the competition for processor time. If processes can be merged to perform similar processing on multiple channels in parallel by one task, the delay times are not much different from the delay times for one channel alone.

Real-time response depends on being fast enough, not just on being fast. We have consistently discussed worst-case timing and meeting response deadlines under loaded operating conditions. Much of what is called "real time on your PC" is in fact concurrent processing with unspecified delays that can be a factor of 100 to 1000 longer than delays that you might typically experience on a Data Acquisition Processor. The Data Acquisition Processor is not a replacement for a workstation system but rather an extension to it. Even so, there are limits, and response times of Data Acquisition Processors are either sufficient for the application or they aren't. If you can't meet the real-time deadlines, the platform isn't the right one and you will have to consider a higher performance, higher complexity solution.

Data Acquisition Processors are best suited for those applications with special requirements for operating speed, multiple channels, novel or complex control algorithms, or supporting computations. What seems like an expensive solution at first, after applying the solution to multiple channels and incorporating important monitoring, testing, and logging features, can turn out to be a very competitive solution. The flexibility of the high-level development environment that does not require complex and specialized embedded system tools means a low-cost infrastructure for supporting development. Effort can be concentrated on the control problem, rather than on making the hardware components work together. Methods that might ordinarily be unsupportable can become feasible, opening a new range of opportunities for process improvements and long-term cost savings.