



DeltaV Control Module Execution

This document explains how the execution of control modules is managed in the DeltaV controller.

© Emerson Process Management 1996—2007 All rights reserved.

DeltaV, the DeltaV design, SureService, the SureService design, SureNet, the SureNet design, and PlantWeb are marks of one of the Emerson Process Management group of companies. All other marks are property of their respective owners. The contents of this publication are presented for informational purposes only, and while every effort has been made to ensure their accuracy, they are not to be construed as warranties or guarantees, express or implied, regarding the products or services described herein or their use or applicability. All sales are governed by our terms and conditions, which are available on request. We reserve the right to modify or improve the design or specification of such products at any time without notice.





Contents

Introduction.....	4
Controller Task Categories.....	4
Module Priorities.....	7
Controller Loading Recommendations	8



Figures

Figure 1. Relative controller task priorities	5
Figure 2. Module priority determination.....	7



Introduction

The DeltaV controller is responsible for executing a variety of tasks. The most obvious one is the execution of the configured control modules. However, the controller performs several other tasks, such as I/O processing, communications, and diagnostics.

All of the above tasks have to be serviced within a certain time for the controller to function optimally. A scheduler is responsible for controlling the execution of the modules in the controller. The controller has a finite amount of resources (CPU time and memory) to execute its tasks. This document explains how the execution of control modules is managed.

Controller Task Categories

The breakdown of all the tasks in the DeltaV controller is beyond the scope of this document. However, it is useful to generalize the tasks into two major categories to develop an understanding of the relative priority of those categories. The two main categories are CONTROL tasks and OTHER tasks. The main tasks in the OTHER category are redundancy management, communications, and self-test. (See Fig.1 for the relative priority of these tasks.) There is no difference in the way the scheduler works on an MD/MDPlus compared to an M5Plus controller. The MD has a faster CPU and a floating point co-processor. This allows it to execute more instructions (especially floating point) per second.

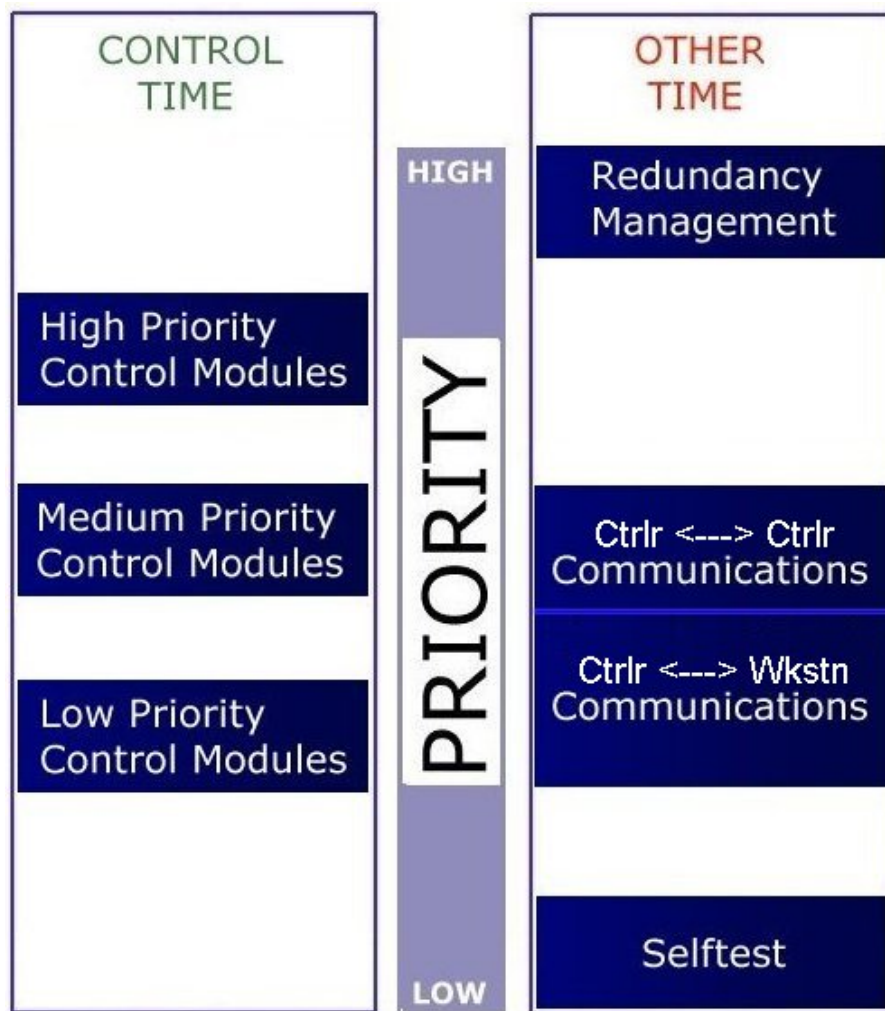
If the controller is normally loaded, then all these tasks can be serviced effectively, and the priority of the tasks is not called into play. However, it is possible to overload the controller by requiring it to perform too much configuration and communication processing. When the controller is overloaded, lower priority tasks are paused until higher priority ones can be completed. Typically there are two ways that a controller can be overloaded :

1) The maximum amount of CONTROLtime has been reached.

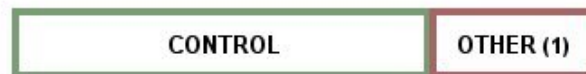
There is a fixed maximum amount of CPU time that is allocated to run the control modules. This is known as CONTROLtime. When there is no configuration in the controller, actual CONTROLtime is 0% and the diagnostic attribute FreTim (Free Time) reads 100%. If the configuration loaded in the controller needs to use more than the allotted CONTROLtime, the execution of one or more modules will be slower than the configured scan rate. This condition is called 'Slippage'.

2) Too much communications processing

If the configuration does not consume all of the CONTROLtime then there is $(100\% - \text{actual CONTROLtime})$ left for the OTHER operations. If the execution of the OTHER tasks requires more than $(100\% - \text{actual CONTROLtime})$, controller operations will systematically degrade by making lower priority operations in the controller wait until a higher priority operation has completed. In very severe overload situations, some tasks in the controller will suspend indefinitely until the controller has been re-configured to reduce load.



Controller time Allocation



(1) OTHER time is allowed to float if
actualCONTROL time < maximumCONTROL time

Figure 1. Relative controller task priorities

Controller-to-controller communications are conducted via request/response messages. Most controller-to-workstation communications (reporting of data) are conducted using unsolicited messages. Processing of request/response messages is a higher priority task than unsolicited messages. This means that, in general, controller-to-controller communications is not affected by controller-to-workstation communications.



In the common overload scenario, module execution will slip and the slippage indicators will indicate which priority modules are slipping. In this overload scenario, the scheduler is still able to police module execution.

However, in certain overload conditions, the execution of Medium and Low priority control modules sometimes has to time-slice with communications tasks. This is normally caused by a misbehaving configuration that makes the scheduler lose its ability to fully police the execution of modules. This typically occurs when a module takes longer than 50ms to execute. This can happen if there are too many heavy function blocks (such as calc-logic) in one module. In this case, the scheduler is inhibited from being able to enforce the CONTROLtime limit for module execution. Since it is important that communications can still happen during this condition, operating system time slicing allows the communications tasks to get some share of the processor, despite the mis-behaving control module that is trying to hog CPU time. It is important to ensure that the communications with the controller are still alive in this situation because the fault can be rectified with a download of a more suitable configuration.

I/O scanning happens in an interrupt service routine and is therefore not subject to task priorities. As far as controller loading is concerned, the time spent scanning I/O is negligible compared with the major task categories. Most of the I/O processing occurs in the context of control, for example when an I/O function block is executed.

If the controller does not have enough time to execute all tasks, then the major pieces of functionality to degrade will be certain lower priority communication tasks and Low Priority Control. The initial degree of degradation causes the reporting of data to the workstations to be sluggish. For Low Priority Control, modules will not execute on time. In an extreme case, data is not reported to the workstation at all and Low Priority modules do not execute.



Module Priorities

There are three module priorities : High, Medium, and Low. The priority of a module is determined by its configured scan rate.

Scan Rate (seconds)	High Priority	Medium Priority	Low Priority
0.1	✓		
0.2	✓		
0.5		✓	
1		✓	
2		✓	
5		✓	
10			✓
30			✓
60			✓

Figure 2. Module priority

The allocation of the available CPU time to the three priority groups is dynamic, within certain bounds. High priority control is not allowed to take the entire amount of CONTROLtime. High priority modules will start slipping if total high priority control exceeds the limit (45% of CONTROLtime) set for High priority control. This guarantees that there is some time for medium and low priority modules to execute. Hence, it is possible to see slippage of High priority modules even when there is some overall free time in the controller. If the modules cannot complete in the time allotted, there will be slippage. When this happens one or more of the diagnostic attributes HPCTONTIME, MPCTONTIME, LPCTONTIME will read less than 100%. Assuming that there is some free time, all modules will eventually execute.

The controller automatically adjusts the priority of a module when it is placed in the debug state. In v5.x, High and Medium priority modules run at Medium priority and Low priority modules run at Low priority. In v4.2 all debug sessions run at Medium Priority.

The limits for CONTROLtime and OTHER activities in the controller are release dependant and may be changed to make optimum use of the controller resources.



Controller Loading Recommendations

- The function block scan rate multiplier should be used to make some function blocks run slower than the configured module scan rate if these blocks do not have to execute on every scan of the module.
- Use the lowest practical scan rate for modules. Many process applications only need a 1 sec scan time. Level loops are often very much slower. Temperature loops may be slow too.
- Pay attention to the function block execution order since this will affect the actual response time of a module to a change in the input.
- Be aware of the relative scan rates of interacting modules to achieve the optimum response time. The execution order of modules cannot be configured and no particular order should be assumed.
- Use CALC blocks sparingly because the processing of structured text (expressions) creates a higher loading than an equivalent function block would.
- Verify, via Diagnostics, that there are no unresolved references in the configuration. Unresolved references will create unnecessary burst loads because the controller will periodically try to resolve them.
- In some instances the 'Comparator' function block may be substituted for a 'Calc' / 'Condition' block . The 'Comparator' block has less flexibility than the condition block but it is more efficient.
- It is recommended that a minimum of 10-20% free time be maintained.
- Do not create modules with lots of function blocks, especially CALC blocks.
- In typical configurations, redundancy management consumes (5-15%) of the CPU, depending on the amount of data that has to be transferred to the standby controller. Keep this in mind when planning the configuration.
- Use the **Load Estimator** utility to plan the configuration.