GEH-6827

Mark* VIe Controller Multi-core Capability User Guide

Dec 2019

Public Information

These instructions do not purport to cover all details or variations in equipment, nor to provide for every possible contingency to be met during installation, operation, and maintenance. The information is supplied for informational purposes only, and GE makes no warranty as to the accuracy of the information included herein. Changes, modifications, and/or improvements to equipment and specifications are made periodically and these changes may or may not be reflected herein. It is understood that GE may make changes, modifications, or improvements to the equipment referenced herein or to the document itself at any time. This document is intended for trained personnel familiar with the GE products referenced herein.

GE may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not provide any license whatsoever to any of these patents.

***Public Information** – This document contains non-sensitive information approved for public disclosure.*

GE provides the following document and the information included therein as is and without warranty of any kind, expressed or implied, including but not limited to any implied statutory warranty of merchantability or fitness for particular purpose.

For further assistance or technical information, contact the nearest GE Sales or Service Office, or an authorized GE Sales Representative.

Issued: Dec 2019

We would appreciate your feedback about our documentation.
Please send comments or suggestions to *controls.doc@ge.com*

Safety Symbol Legend

Warning

Indicates a procedure or condition that, if not strictly observed, could result in personal injury or death.

Caution

Indicates a procedure or condition that, if not strictly observed, could result in damage to or destruction of equipment.

Attention

Indicates a procedure or condition that should be strictly followed to improve these applications.

Contents

1 Overview

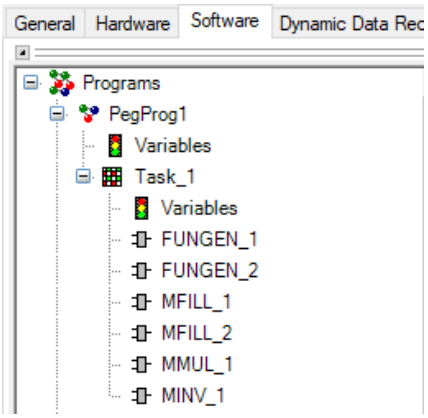
The Mark* VIe controller multi-core capability allows users to distribute their application program to run across multiple CPU cores. The multi-core capability was introduced in ControlST V07.07.00C (Mark VIe V06.09.00C).

Note The controllers that support the multi-core capability are identified in the *Mark VIe and Mark VIeS Control Systems Volume II: System Guide for General-purpose Applications* (GEH-6721_Vol_II), the chapter *Controllers*.

The ToolboxST* application (installed with the ControlST* Software Suite) is used to configure the controller for multi-core capability.

This document assumes that users are familiar with creating user application programs in ToolboxST for the Mark VIe controller by defining the following properties:

- **Frame Period** for the controller
- **Programs** and their execution order
  - **Variables** within a Program, including scope (Global, Member, Local)
  - **Tasks** within a Program, including execution order, frame multiplier, and frame offset
    - **Variables** within a Task, including scope (Global, Member, Local)
    - **Blocks** within a Task, including execution order and interconnections



ToolboxST User-defined Application Program Hierarchy

Note For further details about the ToolboxST user application program hierarchy, refer to the *ToolboxST User Guide for Mark Controls Platform* (GEH-6700 or GEH-6703), the section *Software Configuration Hierarchy.*

The multi-core capability extends this programming paradigm to allow Programs to run on multiple CPU cores. The number of available cores that can be configured to use the multi-core capability is defined by the specific controller platform. For example, the UCSDH1 controller platform provides two CPU cores.

Additionally, the multi-core capability allows Programs to run at different execution rates. On a given CPU core, a faster execution rate program will preempt the execution of the slower execution rate programs. The number of available execution rates on a given CPU core is defined by the specific controller platform. For example, the UCSDH1 supports up to four execution rates on a CPU core.

This document discusses the configuration, program execution, and application use of the controller multi-core capability, as well as the restrictions and limitations. Troubleshooting information is also provided.

Notes

2 Configuration

The following terms are used in reference to multi-core user application program execution in the ToolboxST Mark VIe component:

- **Execution Group** is a collection of Programs that are assigned to run on the same CPU core and at the same execution rate, where:
  - **Core** specifies the CPU core for the Execution Group.
  - **Frame Multiplier** determines the execution rate for the Execution Group by specifying that it runs at an integer multiple of the controller's base execution rate.

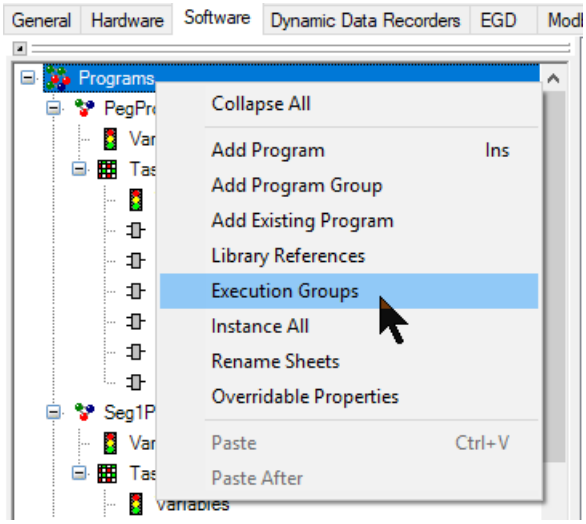
Note A *Program Group* is not the same as an *Execution Group*. A Program Group is simply a container used to logically group Programs.

By default, ToolboxST defines one Execution Group as the ***Primary Execution Group (PEG)***. The PEG is configured to run on Core 0 at the controller's base execution rate (Frame Multiplier = 1). The configuration of the PEG cannot be modified by the user.

Additional Execution Groups, called ***Secondary Execution Groups (SEG)***, can be created and modified by the user. Users create SEGs in ToolboxST.

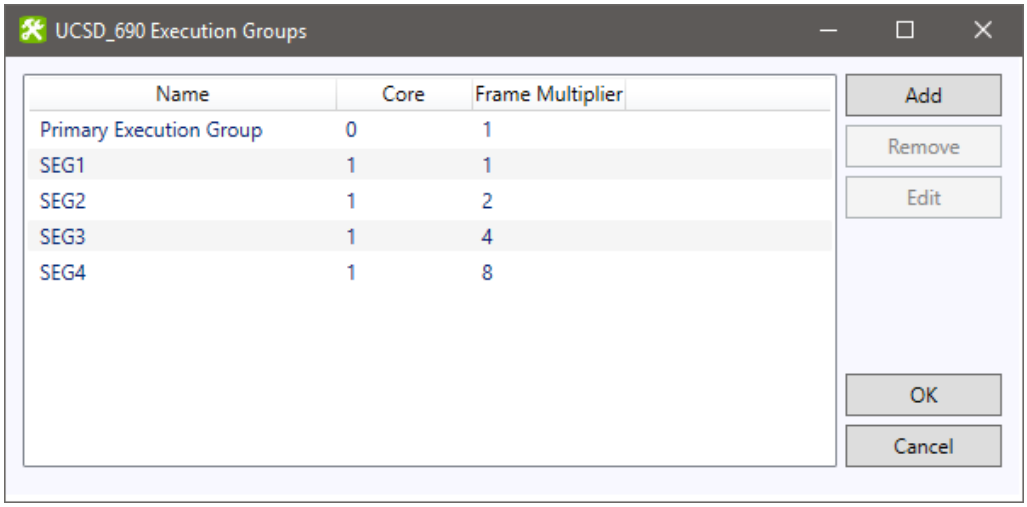
➢ To create a SEG in ToolboxST

1. From the Mark VIe **Component Editor**, click the **Software** tab, right-click **Programs** in the Tree View, and select **Execution Groups**.



ToolboxST Execution Groups Menu Item

An *Execution Groups* window displays a list of Execution Groups from which users can view, add, remove, or edit Execution Groups. The following figure displays an example list of Execution Groups in ToolboxST.

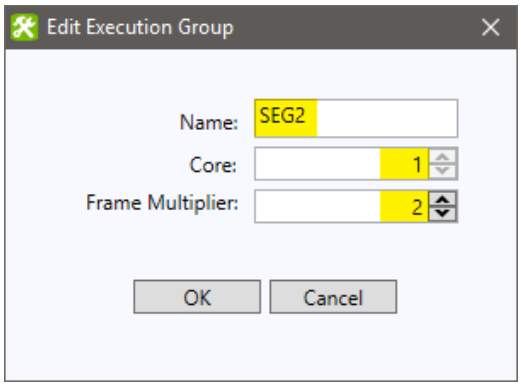


Example List of Configured Execution Groups

This example shows a Primary Execution Group (PEG) and four user-created Secondary Execution Groups (SEG) named SEG1, SEG2, SEG3, and SEG4, configured as follows:

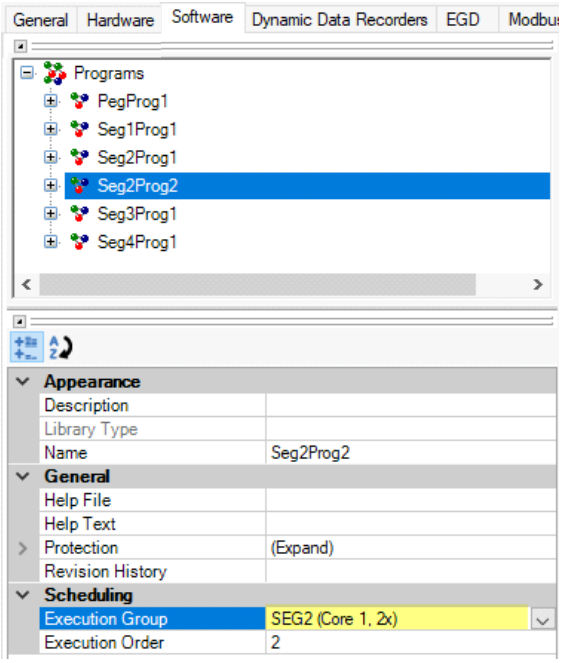
- PEG – executes on Core 0 at the controller's base execution rate.
- SEG1 – executes on Core 1 at a rate equal to the controller's base execution rate.
- SEG2 – executes on Core 1 at a rate 2 x slower than the controller's base execution rate.
- SEG3 – executes on Core 1 at a rate 4 x slower than the controller's base execution rate.
- SEG4 – executes on Core 1 at a rate 8 x slower than the controller's base execution rate.

2. Click **Add** to add a new SEG.
3. Provide a **Name** for the group, select the **Core** and **Frame Multiplier** values, then click **OK** to display the newly created SEG in the list of Execution Groups.



Add or Edit Secondary Execution Group

Programs are assigned to Execution Groups using the Scheduling property, Execution Group, as shown in the following figure.



Execution Group Property

In this example, the Program named **Seg2Prog2** is assigned to the Secondary Execution Group, SEG2, which causes it to execute on Core 1 at a rate that is two times slower than the controller's base execution rate, per the definition of SEG2.

Multiple Programs can be assigned to a given Execution Group. All new Programs created by the user are automatically assigned to the PEG. The user can change a Program's Execution Group to any Execution Group that has been previously defined.

The following modifications to the multi-core configuration will cause a **major** controller equality difference, which requires an offline download (controller reboot):

- Add or delete a SEG
- Change the Core value of a SEG
- Change the Frame Multiplier value of a SEG
- Change the Execution Group of a Program
- Change the Writing Execution Group of a variable

Any other modifications to the multi-core configuration will cause a **minor** controller equality difference, which requires an online download only (no controller reboot). Some examples are:

- Change the name of a SEG
- Change the execution order of a Program, without changing its Execution Group
- Add or remove Programs from an Execution Group

Restrictions

The following restrictions apply to SEG configuration:

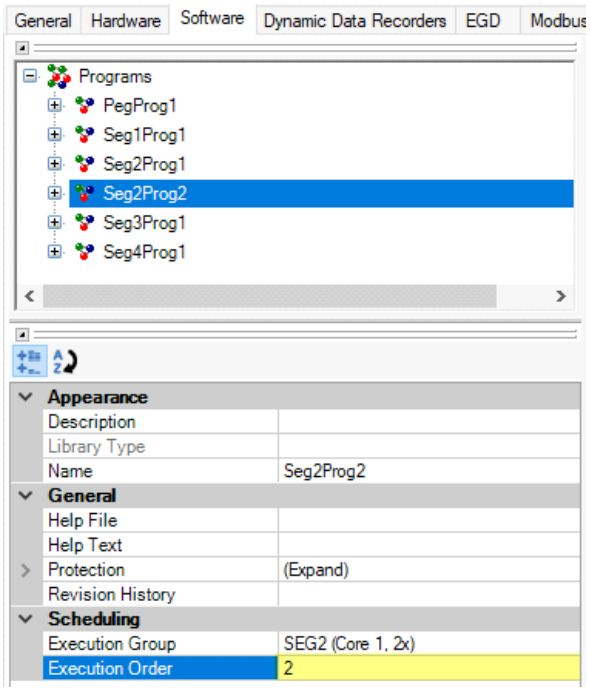
- SEGs cannot be assigned to Core 0; this core is reserved for the Primary Execution Group (PEG).
- All SEGs assigned to the same Core must have unique Frame Multipliers.
- All SEGs assigned to the same Core must have Frame Multipliers that are integer multiples of each other.
- There is a limit to the number of SEGs that can be assigned to the same Core. This limit is platform dependent. For example, on the UCSDH1, the limit is four.

Notes

3 Program Execution

All Programs assigned to a given Execution Group will execute sequentially within that Execution Group in the order specified by the Program's Scheduling property, Execution Order, as shown in the following figure.

Note In this document, a reference to an Execution Group is actually referring to the collection of all Programs assigned to that Execution Group.



Execution Order Property

In this example, both **Seg2Prog1** and **Seg2Prog2** Programs are assigned to the SEG2 Execution Group. The **Seg2Prog1** Program will execute first (Execution Order = 1) and the **Seg2Prog2** Program will execute second (Execution Order = 2).

The following rules dictate how Secondary Execution Groups (SEG) execute on the controller:

- All SEGs execute phase-synchronous to the Primary Execution Group (PEG), by aligning common execution frame boundaries.
- All SEGs configured to execute on the same core will execute in a priority-based, preemptive manner. Execution Groups with faster execution rates (those with smaller frame multipliers) will execute at a higher priority and will therefore preempt/block Execution Groups with slower execution rates (those with larger frame multipliers) that execute at a lower priority.

The ToolboxST ***Frame Timeline Profiler*** provides a graphic representation of how Execution Groups execute on the controller. The Frame Timeline Profiler is accessed from the ToolboxST Mark VIe Component View menu.

Note To use the Frame Timeline Profiler, profiling must be enabled for the controller by setting the controller General configuration property, Profiler Enabled, to True. For more information, refer to the *ToolboxST User Guide for Mark Controls Platform* (GEH-6700 or GEH-6703), the section *Frame Timeline Profiler*.

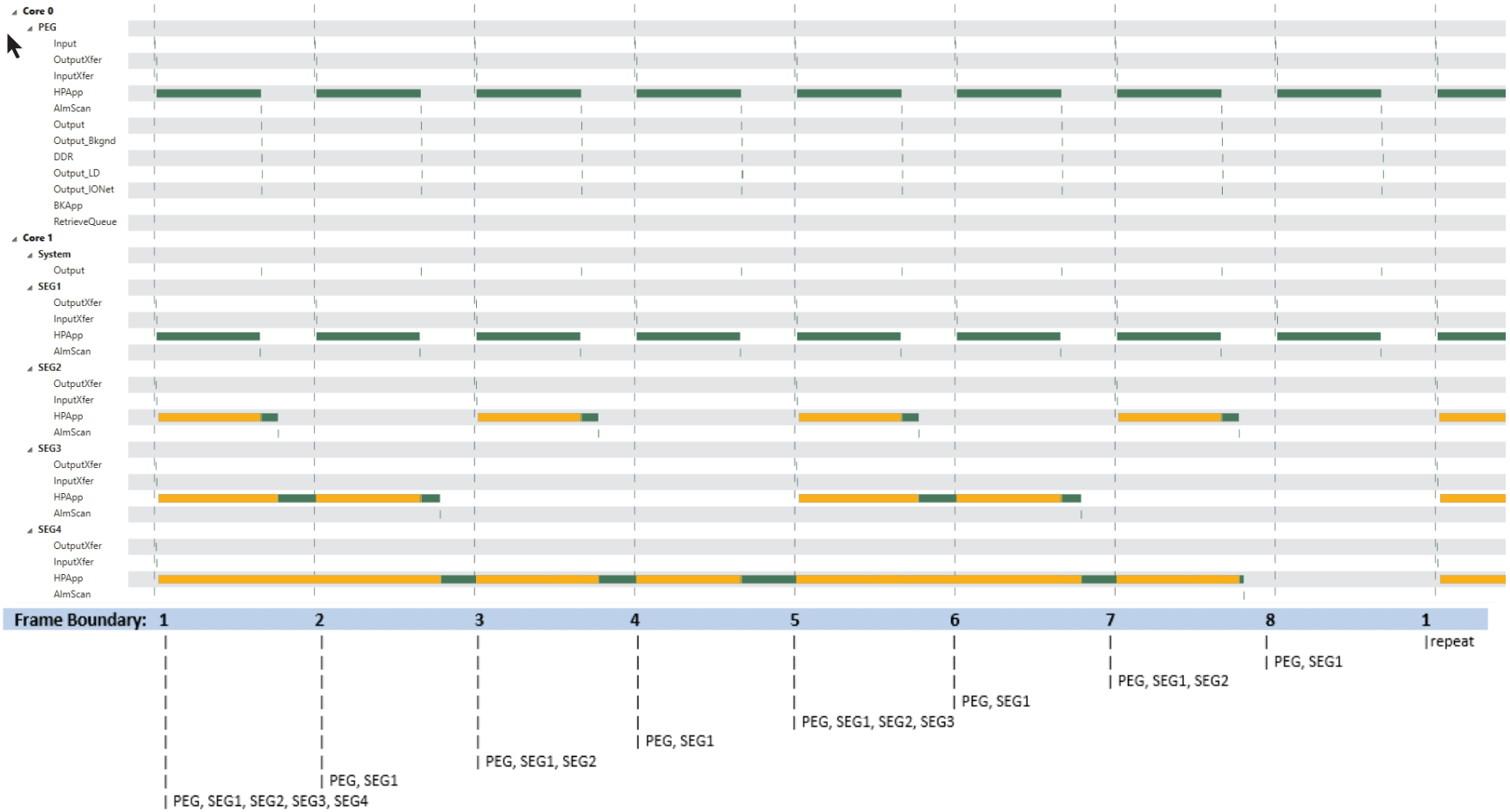
The execution of the PEG and all SEGs is represented on the Frame Timeline Profiler as follows:

- The horizontal axis represents time.
- The vertical axis describes the processor activity grouped by the CPU core and Execution Group.
- The rows designated **HPApp** represent the user application for each Execution Group:
  - Green indicates when HPApp is executing.
  - Orange indicates when HPApp is preempted/blocked by an HPApp running in a higher priority Execution Group.

Using the controller configuration described in the chapter *Configuration*, HPApp execution and preemption is illustrated in the following figure, *Execution Group Frame Boundaries and HPApp Execution/Preemption*.

In this figure, Execution Groups execute as follows:

- PEG Execution Group
  - Executes on Core 0 with Frame Multiplier = 1.
  - Execution starts on frame boundaries 1, 2, 3, 4, 5, 6, 7, 8.
  - Cannot preempt SEG1, SEG2, SEG3, or SEG4 since they are running on a different core.
- SEG1 Execution Group
  - Executes on Core 1 with Frame Multiplier = 1.
  - Execution starts on frame boundaries 1, 2, 3, 4, 5, 6, 7, 8 (every PEG frame boundary).
  - Executes at highest priority on Core 1 so it cannot be preempted.
- SEG2 Execution Group
  - Executes on Core 1 with Frame Multiplier = 2.
  - Execution starts on frame boundaries 1, 3, 5, 7 (every 2nd PEG frame boundary).
  - Executes at 2nd highest priority on Core 1 so it can be preempted by SEG1.
- SEG3 Execution Group
  - Executes on Core 1 with Frame Multiplier = 4.
  - Execution starts on frame boundaries 1, 5 (every 4th PEG frame boundary).
  - Executes at 3rd highest priority on Core 1 so it can be preempted by SEG1 and SEG2.
- SEG4 Execution Group
  - Executes on Core 1 with Frame Multiplier = 8.
  - Execution starts on frame boundary 1 (every 8th PEG frame boundary).
  - Executes at lowest priority on Core 1 so it can be preempted by SEG1, SEG2 and SEG3.



Legend

(Green): HPApp is executing.

(Orange): HPApp is preempted/blocked by an HPApp running in a higher priority execution group.

Execution Group Frame Boundaries and HPApp Execution/Preemption

Notes

4 Frame Idle Time

Frame Idle Time (%) is a measurement of how much of the PEG's frame is not consumed by executing an HPApp. The definition of Frame Idle Time (%) for the PEG is not affected by the introduction of SEGs because SEGs do not run on the same core as the PEG.

Frame Idle Time (%) for a SEG is more complicated in that there can be more than one SEG, running at different priorities, on the same core. In this situation, the amount of a SEG's frame that is not consumed by executing that group's HPApp may not be fully available due to possible preemption by higher priority SEGs on the same core.

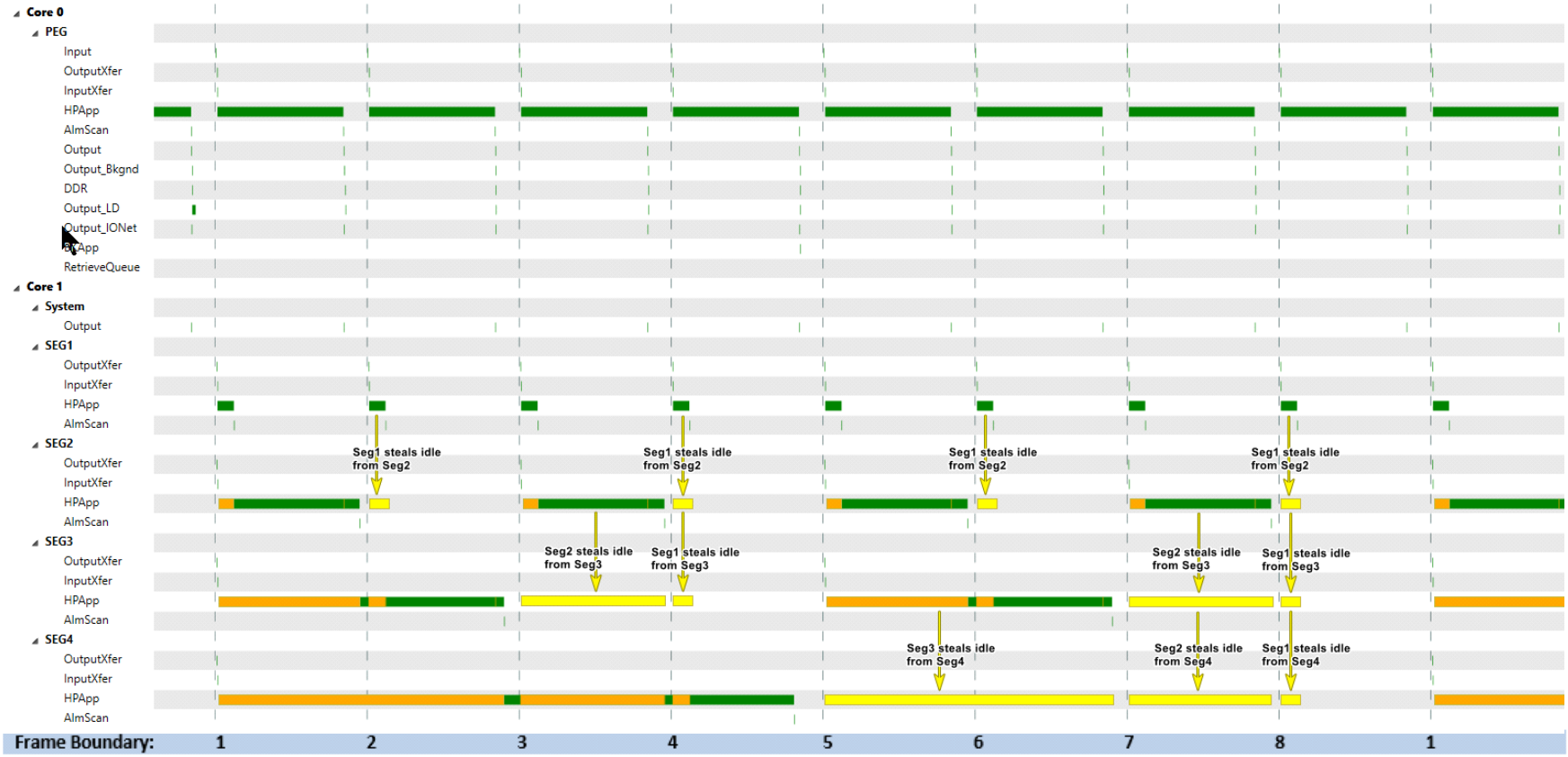
Using the controller configuration described in the chapter *Configuration*, frame idle preemption is illustrated in the following figure, *Execution Group Frame Idle Time*.

In this figure, yellow bars identify the portions of each SEG frame's idle time that are not available to run an additional HPApp workload because execution in that region would be preempted by an HPApp running in a higher priority SEG on the same core. This is described for each Secondary Execution Group as follows:

- SEG1 Execution Group
  - Idle time is not reduced since this is the highest priority Execution Group on the core.
- SEG2 Execution Group
  - Idle time is reduced by SEG1 HPApp running in frames 2, 4, 6, 8.
- SEG3 Execution Group
  - Idle time is reduced by SEG2 HPApp running in frames 3, 7.
  - Idle time is reduced by SEG1 HPApp running in frames 2, 4, 6, 8.
- SEG4 Execution Group
  - Idle time is reduced by SEG3 HPApp running in frames 5, 6.
  - Idle time is reduced by SEG2 HPApp running in frame 7.
  - Idle time is reduced by SEG1 HPApp running in frame 8.

From this example, it is obvious that adding workload to a higher priority SEG will reduce Frame Idle Time (%) not only in that Execution Group, but also in all lower priority SEGs, by both of the following mechanisms:

- Preempting execution of an HPApp in the lower priority SEG
- Reducing idle time in the lower priority SEG



Legend

(Green): HPApp is executing.

(Orange): HPApp is preempted/blocked by an HPApp running in a higher priority execution group.

(Yellow): Portions in each SEG frame that are not available to run additional HPApp.

Execution Group Frame Idle Time

5 Variable Access

Access to variables is restricted in ToolboxST as follows:

- Variables defined with **Member** or **Local** scope cannot be accessed (Read or Write) outside of the defining Execution Group.
- Variables defined with **Global** scope can be Read-accessed by any Execution Group.
- Variables defined with **Global** scope can be Write-accessed by only one Execution Group, recommended (but not required) to be the defining Execution Group.

For a Global variable that is accessed by multiple Execution Groups, variable coherency is ensured in the context of the execution frame of each accessing Execution Group. To make this possible, a single user-defined Global variable results in multiple copies of the variable in the Mark VIe controller, one for each accessing Execution Group. ToolboxST automatically creates configuration records for the Mark VIe controller that define each copy of the variable and describe how the copies are accessed by the Execution Groups so that the Mark VIe controller can provide the appropriate synchronization of the copies.

Based on the configuration records that ToolboxST provides, the Mark VIe controller provides coherent and synchronous data passed from the Writing Execution Group to the Reading Execution Groups as follows:

- For two Execution Groups with frame periods that are integer multiples of each other (such as 20 ms to 100 ms), data is passed directly from the Writing Execution Group to the Reading Execution Group on the first frame boundary common to both (sync boundary).
- For two Execution Groups with frame periods that are not integer multiples of each other (such as 40 ms to 100 ms), data is passed indirectly from the Writing Execution Group to the Reading Execution Group, through the PEG. Data passes from the Writing Execution Group to the PEG on the first frame boundary common to both (sync boundary 1), then from the PEG to the Reading Execution Group on the first frame boundary common to both (sync boundary 2).

Note It is possible to configure Execution Groups with frame periods that are not integer multiples of each other, but only if the controller platform provides more than two CPU cores. For example:

• PEG on Core 0 is 10 ms

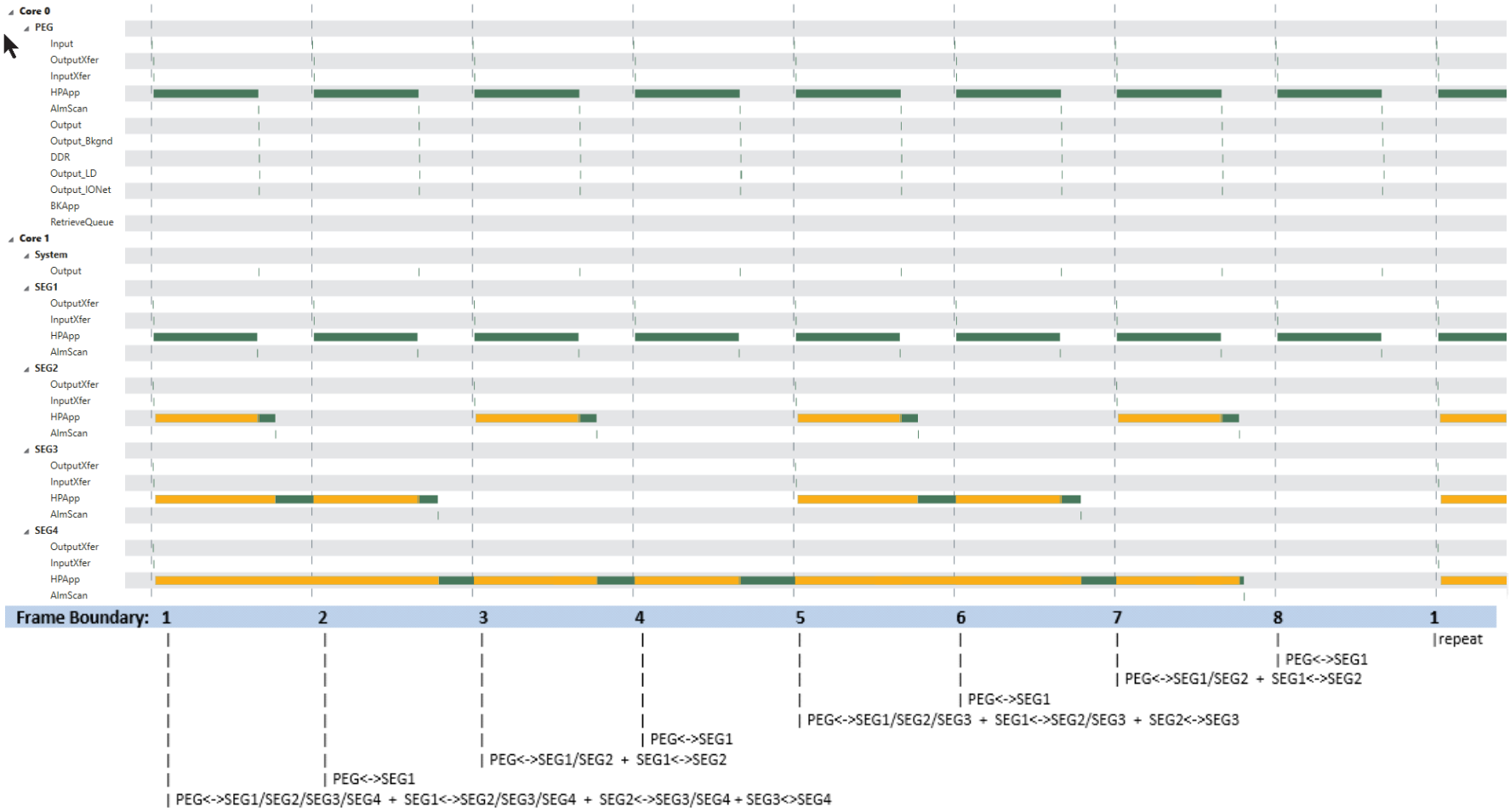
• SEG on Core 1 is 40 ms (Frame Multiplier = 4)

• SEG on Core 2 is 100 ms (Frame Multiplier = 10)

Using the controller configuration described in the chapter *Configuration*, the data passing boundaries between Execution Groups is illustrated in the following figure, *Execution Group Data Passing*.

In this figure, the data passing that occurs at each frame boundary is identified, and is described as follows:

- PEG <-> SEG1 data pass at frame boundaries 1, 2, 3, 4, 5, 6, 7, 8
- PEG <-> SEG2 data pass at frame boundaries 1, 3, 5, 7
- PEG <-> SEG3 data pass at frame boundaries 1, 5
- PEG <-> SEG4 data pass at frame boundary 1
- SEG1 <-> SEG2 data pass at frame boundaries 1, 3, 5, 7
- SEG1 <-> SEG3 data pass at frame boundaries 1, 5
- SEG1 <-> SEG4 data pass at frame boundary 1
- SEG2 <-> SEG3 data pass at frame boundaries 1, 5
- SEG2 <-> SEG4 data pass at frame boundary 1
- SEG3 <-> SEG4 data pass at frame boundary 1



Legend

(Green): HPApp is executing.

(Orange): HPApp is preempted/blocked by an HPApp running in a higher priority execution group.

Execution Group Data Passing

Using the controller configuration described in the chapter *Configuration*, the implications of data passing between Execution Groups is illustrated in the following figure.

Note In the use-case illustrated in the following Trend, PEG is the Writing Execution Group and SEG1, SEG2, SEG3, and SEG4 are the Reading Execution Groups.

Note Sampling of data in the Trender is a best effort synchronous to the PEG and asynchronous to any SEG.

Legend

(Black): PEG sample
(Red): SEG1 sample
(Blue): SEG2 sample
(Magenta): SEG3 sample
(Green): SEG4 sample

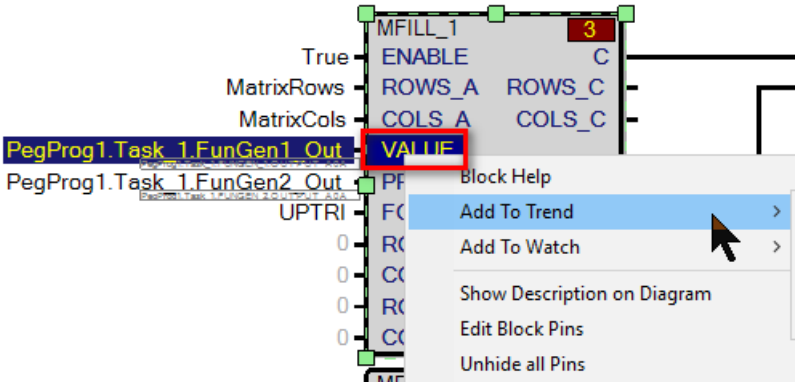
Trend Capture of Execution Group Data Passing

In this Trend, the variable **VALUE** is sampled in the context of the Writing Execution Group (which is PEG) and each Reading Execution Group (which are SEG1, SEG2, SE3 and SEG4). The data passing that occurs for **VALUE** at each frame boundary is described as follows:

- PEG sample (black line)
  - Value updates at frame boundaries 1, 2, 3, 4, 5, 6, 7, 8 to the value calculated by PEG
  - Value is held for one PEG frame, since PEG Frame Multiplier = 1
- SEG1 sample (red line)
  - Value updates at frame boundaries 1, 2, 3, 4, 5, 6, 7, 8 to the value calculated by PEG in the previous PEG frame, causing a one PEG frame delay
  - Value is held for one PEG frame, since SEG1 Frame Multiplier = 1
- SEG2 sample (blue line)
  - Value updates at frame boundaries 1, 3, 5, 7 to the value calculated by PEG in the previous PEG frame, causing a one PEG frame delay
  - Value is held for two PEG frames, since SEG2 Frame Multiplier = 2
- SEG3 sample (magenta line)
  - Value updates at frame boundaries 1, 5 to the value calculated by PEG in the previous PEG frame, causing a one PEG frame delay
  - Value is held for four PEG frames, since SEG3 Frame Multiplier = 4
- SEG4 sample (green line)
  - Value updates at frame boundary 1 to the value calculated by PEG in the previous PEG frame, causing a one PEG frame delay
  - Value is held for eight PEG frames, since SEG4 Frame Multiplier = 8

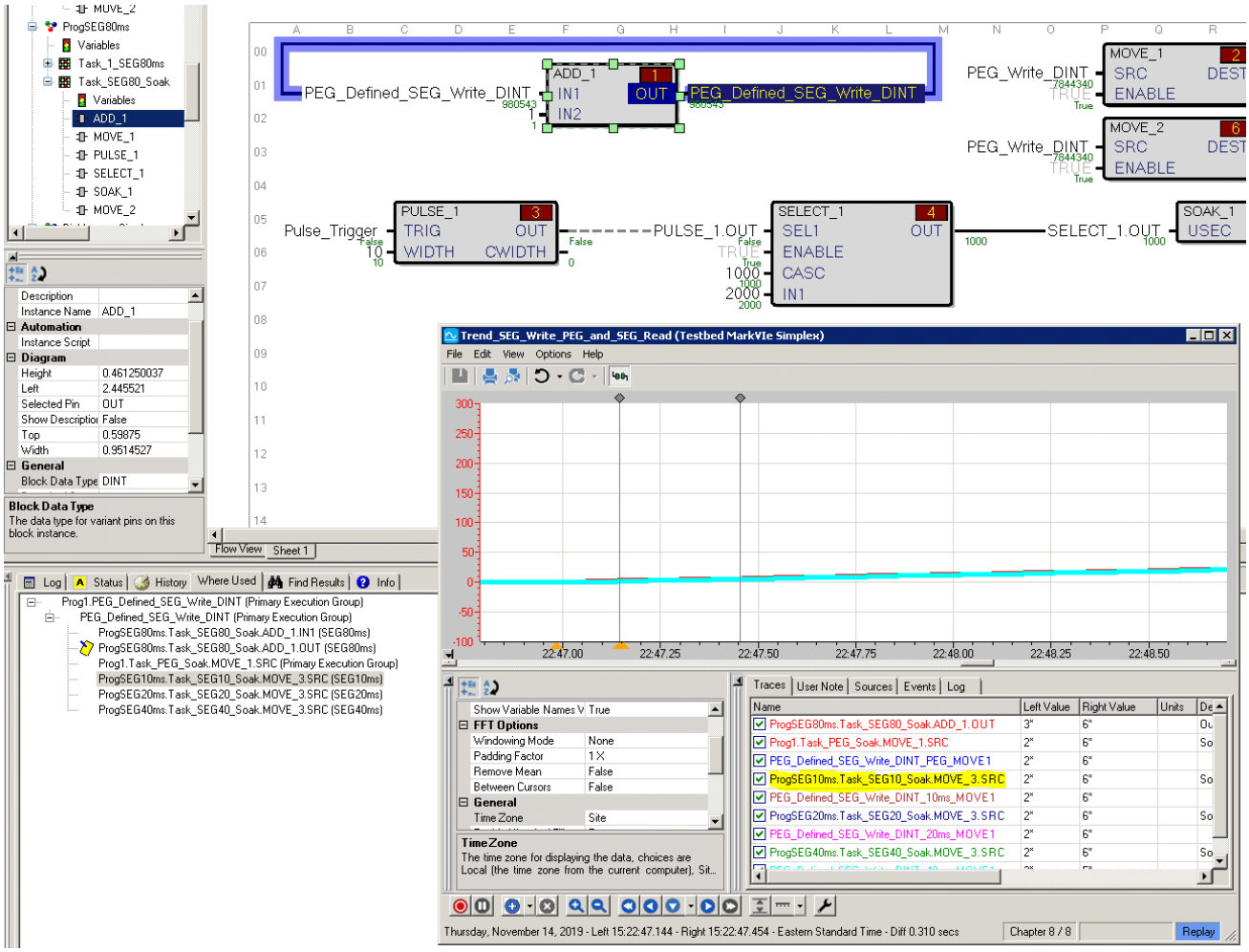
By default, any variable added to a Trend is sampled from the Writing Execution Group. Perform one of the methods provided in the following procedure to add the data-passed version of a variable sampled from a Reading Execution Group into the Trender.

➢ **To add a variable to Trend using right-click method:** from the **Block Diagram Editor**, right-click the connected pin inside a Reading block and select **Add to Trend**.



Add Variable to Trend Using Right-click Method

➢ **To add a variable to Trend using Drag and Drop method:** select the variable, then select the **Where Used** tab to display all instances in the application where the variable is being written, as well as being read by other Execution Groups (the Writing and Reading Execution Groups are listed at the end of the entries and are enclosed in parentheses). Drag a specific usage of the variable from the **Where Used** display and drop it to the Trender window.



Add Variable to Trend Using Drag and Drop Method

Forcing a Global variable that is accessed by multiple Execution Groups can be done in the context of any accessing Execution Group. The force occurs in the Writing Execution Group and the Reading Execution Groups acquire the forced value through the data passing mechanism previously described.

In a redundant control set, state exchange voting for a Global State variable that is accessed by multiple Execution Groups is performed synchronously to the execution frame of the Writing Execution Group. Execution Groups reading the State variable acquire the voted value through the data passing mechanism.

The following properties are supported for variables that are defined in SEGs:

- NVRAM — Saved synchronous to the execution frame of the variable's Writing Execution Group.
- Alarm — Scanned synchronous to the execution frame of the variable's Writing Execution Group.
- Controller collected (Compressed Data Log [CDL]) — Sampled synchronous to the CDL one-second sample rate, which is asynchronous to the execution frame of the variable's Writing Execution Group.

Notes

6 External Interfaces

The Mark VIe runtime executes all external interfaces (external I/O, external data Read/Write) on Core 0 at the controller's base execution rate. Therefore, connecting an external interface to a variable implicitly makes the PEG a Reading and/or Writing Execution Group of that variable. The data passing mechanism as described in the section, *Variable Access*, will pass the variable value to or from any accessing SEGs. No additional latency will be incurred when accessing the variable in a SEG except as required to pass the data at the correct sync boundary.

ToolboxST allows the following external input/output connections to a Global variable defined in any Execution Group:

- IONet I/O
- FOUNDATION Fieldbus™ I/O
- EtherCAT I/O

ToolboxST allows the following external Read connections to a Global variable defined in any Execution Group with the External Access property set to *Read Only*:

- Unit Data Highway (UDH) Ethernet Global Data (EGD)
- WFMS
- Modbus® Slave
- OPC®-UA Data Server

ToolboxST allows the following external Read/Write connections to a Global variable defined in any Execution Group with the External Access property set to *Read/Write*, with the constraint that a connection is not allowed if the Global variable is also written by application code in a SEG:

- UDH EGD
- WFMS
- Modbus Slave
- OPC-UA Data Server

Notes

7 Limitations

The following limitations apply to the multi-core capability:

- ToolboxST will not allow the following Standard Block Library blocks to be used in a SEG:
  - TOTALIZER
  - SYS_OUTPUTS
  - LOGIC_BUILDER
  - LOGIC_BUILDER_SC
- ToolboxST will not allow the following Distributed Control System (DCS) Block Library blocks to be used in a SEG:
  - OVERRIDE
  - DUALSEL
  - DUALSEL_V2
  - MEDSEL
  - MEDSEL_V2
  - QUADSEL
  - QUADSEL_V2
  - M_O_V
  - M_O_V_V2
  - M_O_V_V3
  - M_O_V_V4
  - S_O_V
  - S_O_V_V2
  - S_O_V_V3
  - BREAKER
  - BREAKER_V2
  - GRP
  - GRP_V2
  - GRP_V3
  - STARTER
  - STARTER_V2
  - STARTER_V3
  - M_O_V_JOG
  - M_O_V_JOG_V2
  - M_O_V_JOG_V3
  - PID_MA_ENH
  - PID_MA_ENH_V2
  - OVR_ST_ENH
  - OVR_ST_ENH_V2
- ToolboxST will not allow the following Legacy Block Library blocks to be used in a SEG:
  - OVR_ST
  - PID_MA
- ToolboxST will not allow the following *special tasks* to be used in a SEG:
  - Sequential Function Charts
  - FOUNDATION Fieldbus

Notes

8 Troubleshooting

The following tools can be used for troubleshooting and debugging:

- ToolboxST features such as Block Diagrams, Trender, and Watch Windows can access live data in any Execution Group.
- Capture blocks can be scheduled to run and sample data in any Execution Group.
- Dynamic Data Recorders (DDR) only sample data synchronous to the PEG. If a DDR is configured to sample a variable written in a SEG, the data will be sampled asynchronously to the writing of the variable.
- A Global Variable Report can provide information about variable usage by Execution Groups. Specifically, the following fields can be included in the report and used for sorting and filtering:
  - Number of Execution Groups accessing a variable
  - List of Execution Groups accessing a variable
- ToolboxST Frame Timeline Profiler can be used to visualize execution of Execution Groups in the controller.

Notes