

Avoiding Task Overrun Errors

Question

My model does not run in real-time and stops with overrun errors. What can I do to avoid this? How can I decrease the turnaround time of my model?


Solution

There is no unique solution to avoid overrun situations that applies to all models. But some general tips might help to decrease the turnaround time and to improve the efficiency of the generated code:

- Make use of the multiple timer task mode. Model parts that do not necessarily have to be calculated with the fixed-step size of the model (i.e., the fastest timer-driven task) must be assigned to a slower task.
For more detailed information about this topic, refer to *RTI and RTI-MP Implementation Guide*, chapter *Handling Tasks > Timer Tasks*.
- Avoid using unnecessary blocks, especially if the blocks are fed with vector or matrix signals. For example, a Gain block which is driven by an FFT-signal with 1024 elements results in 1024 multiplications and requires the memory for 1024 double values.
- Avoid using the *Math Function Block*.
For example, do not use the pow function to calculate powers of integer values, but use Product blocks instead.
- Only use the Saturate on integer overflow option if it is really required. This option can be selected for several Simulink blocks (Gain, Sum,...). Turning this option off results in more efficient code.
- Making use of atomic subsystems with the RTW system code option set to Function has an influence on the turnaround time. In general, the actual influence of the option cannot be foreseen. Especially in case of very large models, generating atomic subsystem code into separate functions makes it possible to use a higher optimization level for compilation.

Refer to [FAQ 210](#) for details about how to reduce the model size or complexity.


- As of MATLAB R13, the Reusable Function option exists for atomic subsystems. Using this option might also have a positive influence on the efficiency of the generated code.
- As far as possible, (atomic) subsystems should contain only blocks that are calculated with the same sample-time. As a result, the generated code contains less calls to `ssIsSampleHit` or `rtIsSampleHit`. For more information about this topic, refer to *Real-Time Workshop User's Guide*, chapter *Program Architecture: Model Execution*.
- The Inline Parameters, Signal Storage Reuse, Inline Invariant Signals, Local Block Outputs, Buffer Reuse, Expression Folding and Conditional input branch execution options (since MATLAB R13), which can be found in the Simulation Parameters/Configurations Parameters dialog, might have a positive influence on the turnaround time.

| | |
|---|---|
|  | <p>If Signal storage reuse is activated, some block output signals are missing in the ControlDesk variable browser. For more information, refer to FAQ 019.</p> <p>If Inline Parameters is turned on, only selected block parameters are available in the trace file.</p> <p>If Conditional input branch execution is set, different parts of the application will not be calculated and could lead to misinterpretation of the results</p> |
|---|---|

You will find more information on how to optimize the generated code in the *Real-Time Workshop User's Guide*, chapter *Optimizing a Model for Code Generation*.

Excerpt:

- Run slupdate on old models.
- Directly inline C code S-functions into the generated code by writing a TLC file for the S-function.
- Use a Simulink data type other than double when possible.
- Remove repeated values in lookup table data.
- Use the Merge block to merge the output of function-call subsystems.
- Look-up tables and polynomials: Simulink provides several blocks that allow for the approximation of functions. These include blocks that perform direct, interpolated, and cubic spline look-up table operations, and a polynomial evaluation block. (...) Each type of look-up table block has its own set of options and associated trade-offs.
- You can achieve extensive performance gains on most processors by identifying the portions of your block diagram that are really integer calculations (such as accumulators), and implementing them with integer data types. Floating-point DSP targets are an obvious exception to this rule.
- If your model contains Stateflow blocks, select the Use Strong Data Typing with Simulink I/O checkbox (on the Chart Properties dialog box) on a chart-by-chart basis.

| | |
|---|---|
|  | <p>Some of the techniques described in the Real-Time Workshop User Guide are not supported by RTI:</p> <ul style="list-style-type: none"> • Specifying <code>-DREAL_T=float</code> after <code>make_rtw</code> in the Make command edit field in the Simulation parameters dialog must not be done if the model is built for a dSPACE system. This leads to memory corruptions, because several RTLib functions work only for the double data type. • RTI does not support the Block Reduction option. |
|---|---|

Related FAQs

- [FAQ 210](#) Compilation Problems when using large Models

FAQ Overview

<http://www.dspace.com/go/faq>

Support

To request support, please use the form at <http://www.dspace.com/go/supportrequest>

Updates and Patches

Software updates and patches are available at <http://www.dspace.com/go/patches>.
dSPACE strongly recommends to use the most recent patches for your dSPACE installation.

Important Notice

This document contains proprietary information that is protected by copyright. All rights are reserved. The document may be printed for personal or internal use provided all the proprietary markings are retained on all printed copies. In all other cases, the document must not be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of dSPACE GmbH.

© 2018 by:

dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

This publication and the contents hereof are subject to change without notice.

A list of registered dSPACE trademarks is available at: <http://www.dspace.com/go/Trademarks>