dSPACE NEWS

Best Practices for TargetLink Models

New modeling guidelines for TargetLink, produced jointly with a German OEM, aim to optimize the generation of efficient C code from control algorithms. Version 1.0 has just been released and is available to all TargetLink users. Currently, models often require reworking at the interface between control design and software development. The new guidelines cut out a lot of this extra work and boost development productivity.

- **More productivity in the development process**
- **Seamless transition from control design to software development**
- **Tips on efficient code, MISRA conformity, transparent models, and many others**

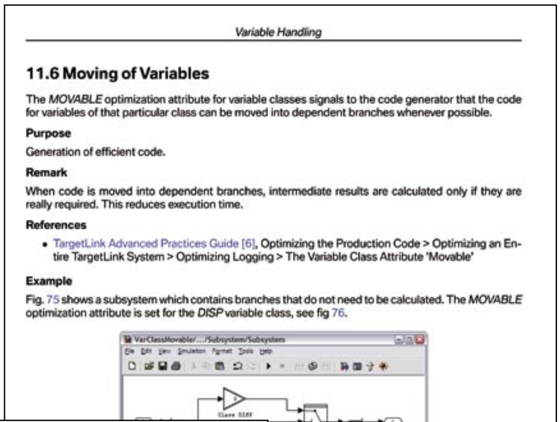
Why Use Modeling Guidelines?

A problem often encountered in target implementation is that the control design includes modeling styles and modeling elements that cannot be transformed into efficient C code. When this happens, models have to be reworked during the software development phase - a time-comsuming and error-prone process. The TargetLink Modeling Guidelines now help control designers to select a suitable language subset in Simulink®/Stateflow®, and software developers to achieve optimum model conversion to highly efficient C code.

Contents of the Modeling Guidelines

The TargetLink Modeling Guidelines comprise around 150 rules, covering the following aspects:

- Transparent controller layout
  Like coding guidelines at software level, modeling rules enhance transparency and readability at model level.
- Suitable language subset
  The defined subset of language elements from MATLAB/Simulink/Stateflow allows optimum implementation by TargetLink.
- Optimum fixed-point code
  The rules provide a guide to converting models into highly efficient fixed-point code, complementing the features (autoscaling, etc.) that TargetLink already contains.
- Code generation options
  The guidelines describe optimization settings for handling variables and functions to generate efficient code.
- MISRA compliance
  The rules help to ensure that the generated code will have maximum compliance with MISRA C.



Variable Handling

11.6 Moving of Variables

The *MOVABLE* optimization attribute for variable classes signals to the code generator that the code for variables of that particular class can be moved into dependent branches whenever possible.

Purpose

Generation of efficient code.

Remark

When code is moved into dependent branches, intermediate results are calculated only if they are really required. This reduces execution time.

References

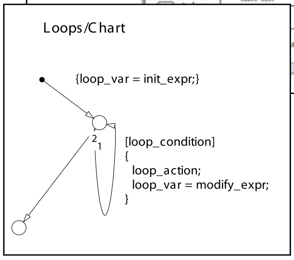
- TargetLink Advanced Practices Guide [6], Optimizing the Production Code > Optimizing an Entire TargetLink System > Optimizing Logging > The Variable Class Attribute 'Movable'

Example

Fig. 75 shows a subsystem which contains branches that do not need to be calculated. The *MOVABLE* optimization attribute is set for the *DISP* variable class, see fig 76.

VarClassMovable/.../Subsystem/Subsystem
File Edit View Simulation Format Tools Help

◀ *Every rule has a name, definition, purpose, background information, and optional references and examples.*



▲ *Using design patterns in Stateflow (modeling a loop in this example) allows direct conversion into efficient C constructs.*

TargetLink users can obtain the Modeling Guidelines (PDF document) free of charge from Technical Sales at *info@dspace.com*