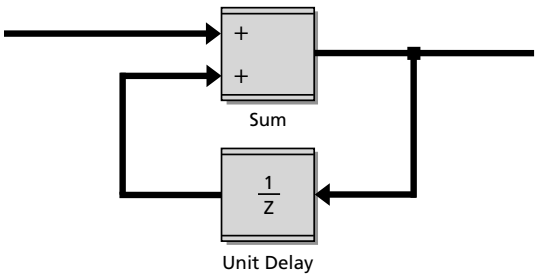
Tracking Down Run-Time Errors

- **Automatic run-time error analysis of TargetLink code**
- **Navigation straight from analyzed code to model**
- **Tool integration ensures high-precision analysis**

Generating the actual function code is not the only way in which TargetLink, the production code generator, helps engineers to develop software for electronic control units. TargetLink users have a wide range of tools for verification and validation at their fingertips – partly in TargetLink itself, and partly in extension solutions like MTest. Another innovative tool has just joined these: the TargetLink-PolySpace integration, which offers protection against run-time errors.

The Sources of Run-Time Errors

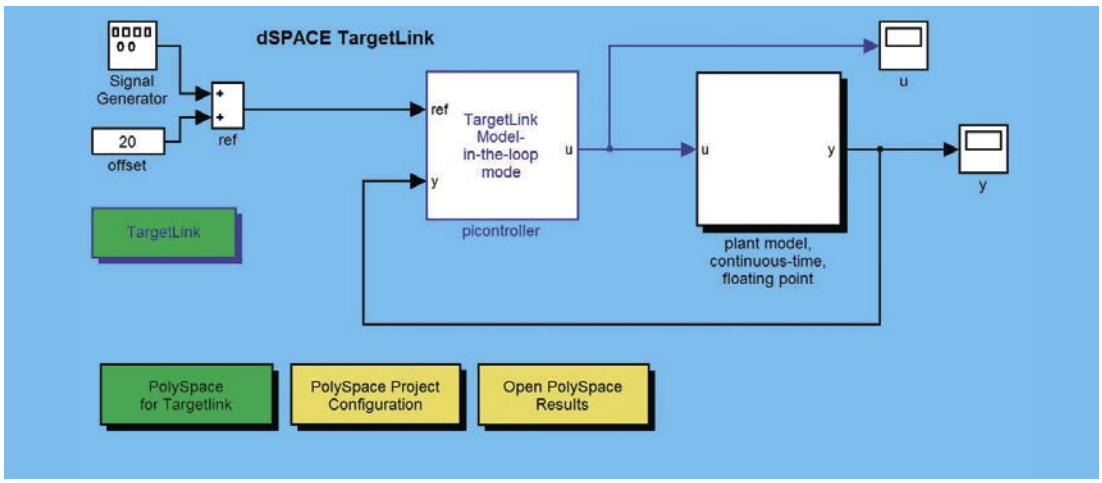
Even though no human programmer can ever attain the degree of correctness of automatic production code generators, the code that these generate is not necessarily free of run-time errors. The reason for this is that errors can creep in during model design, while the function is still being developed. For example, if there is no protection against division by zero or out-of-range values at model level, the code that is generated may contain run-time errors, as any potentially erroneous specification is translated into code 1:1. Run-time errors like these have their sources at model level and should preferably be eliminated at that level. The TargetLink-PolySpace integration greatly simplifies this process.



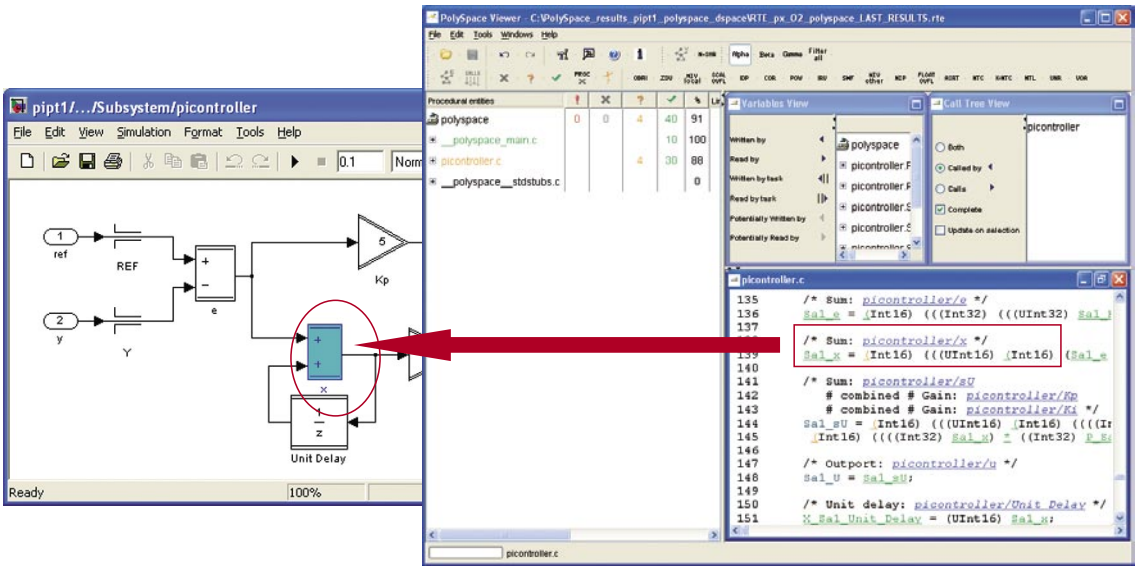
▲ *Model fragment containing feedback that may cause overflow/underflow.*

What Can the TargetLink-PolySpace Integration Do?

Integration means that TargetLink can be directly connected to the PolySpace Verifier, which uses static analysis to analyze the generated code with the aid of what is known as abstract interpretation. This technique returns information on run-time errors with a precision that is comparable with a mathematical proof. Individual code fragments are classified according to whether they will never have any run-time errors, will always have a run-time error, will never be executed (dead code), or may sometimes have a run-time error. Only this last group requires



▶ *Code analysis is configured and run from model level, using additional TargetLink-PolySpace blocks.*



▲ *The PolySpace Viewer uses colors to indicate potential run-time errors in the code and allows navigation straight to the model.*

closer analysis by developers, as these are the cases where, due to the abstractions, the PolySpace Verifier cannot precisely determine whether a run-time error can really occur or not.

Advantages of the TargetLink-PolySpace Integration

Users of both tools benefit greatly from the TargetLink-Polyspace integration in the following ways:

- It takes just a few clicks to run the code analysis from model level. The configuration parameters for the PolySpace Verifier, and the TargetLink subsystem that the code belongs to, are also specified in the model.
- The PolySpace Viewer uses colors to classify the operations in the generated code (green = will never have a run-time error, orange = may sometimes have a run-time error, etc.). Users can navigate straight from the code to the corresponding locations in the model. This makes it easy to trace the critical points back to the model, examine them, and if necessary correct them.
- The precision of the analysis can be greatly enhanced by additional information at model level, such as value range limits for calibratable parameters, and input values. The PolySpace Verifier reads this information from the dSPACE Data Dictionary and uses it in the analysis. This reduces the number of code lines whose run-time behavior cannot be precisely determined.
- PolySpace Verifier explicitly recognizes TargetLink's optimized code generation using the compute-through-overflow technique, which again makes it easier to analyze the generated code.

The tool integration produced by PolySpace and dSPACE not only speeds up the development process, it also smooths the way to verifying the production code that is generated.

Glossary

Dead code –
Fragment of code that can never be executed.

Compute through overflow –
Calculation method for arithmetic operations in which overflows are allowed to occur in intermediate results provided the final result can be given correctly.

Abstract interpretation –
Method of analyzing the semantics of a program, using abstractions to reduce the computation load.

Further information on the TargetLink-PolySpace integration (PolySpace for Model-Based Design) can be obtained from PolySpace at contact@polyspace.com