Verifying High-Integrity Control Software for Mission-Critical Emergency Diesel Generators

By Dr. Jörg Barrho, MTU Friedrichshafen GmbH

In the event of an outage of the primary power supply, power plant operators rely on diesel engine-driven generators for the backup power needed to process vital functions. Nuclear power plant applications require the highest standards of safety and reliability. MTU gensets meet these standards.

To ensure the availability of the generators, MTU diesel engines (Figure 1) use high-integrity control software. Development of this software is based on the IEC 60880 standard. IEC 60880 defines requirements for the software used in computer-based instrumentation and nuclear power plant control systems—specifically, software that performs functions of safety category A, as defined by IEC 61226. Each nation's nuclear regulatory agency may supplement these requirements, and each agency performs an independent audit of control software and the process used to develop it.

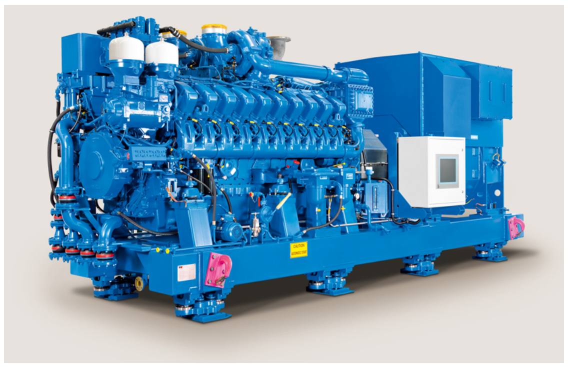


Figure 1. MTU mission-critical diesel genset, used to generate emergency power for a nuclear power plant.

MTU has established a structured development process specifically to meet IEC 60880 requirements. A key part of this workflow is the use of Polyspace® code verifiers to identify and eliminate underflow, overflow, divide-by-zero, memory access, and other run-time errors. Polyspace code verifiers highlight code that is proven to be free of certain categories of run-time errors, enabling the team to focus their reviews on the remaining code.

Shortcomings of Standard Development Processes

When we began developing the control software, we had relatively little in-house expertise in IEC 60880 requirements. Our initial strategy was to use our standard commercial software development process while producing more documentation and conducting additional tests—including Polyspace tests for run-time errors on some of our legacy code.

Polyspace code verifiers highlight each element in the code as green, red, gray, or orange to indicate its status. In our code, a few lines were marked red, meaning they were proven faulty (but justified as correct and valid statements). A high number of elements were colored green, indicating that they were free of run-time errors; however, several orange, or unproven, elements were also highlighted. The detailed results made it easier to identify problems with our code and to justify unproven or unreachable code. However, even the most detailed results are insufficient evidence for regulatory authorities and our customers that the software fulfills the requirements of the standard. We must also show that the processes and tools used to obtain the results are trustworthy.

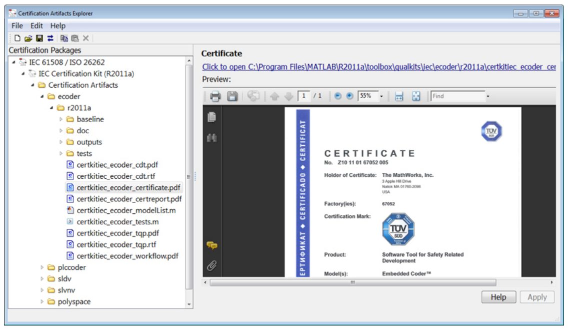
We had to set up a new, highly sophisticated development process that would enable us to satisfy IEC 60880 requirements. Agencies and customers needed proof that we had a plan for software development, that the tools we used were qualified, and that we had followed the plan and used the tools correctly.

Qualifying Polyspace Code Verifiers

A key element of our new software development process was *tool chain management*, an area concerned with the selection and qualification of software development and testing tools. Chapter 14 of IEC 60880 covers the appropriate use of software tools, including those that can increase the integrity of the software development process and improve software reliability. Tools are defined as either *critical* or *noncritical*. A word processor is an example of a noncritical tool. A critical tool is one that may introduce faults into the software (for example, a compiler) or one that may cause a fault in the software to be missed (for example, a verification tool such as Polyspace Client™ for C/C++).

Before we use a critical tool for high-integrity software development, we have to ensure that it is appropriate to the task and that it functions correctly. To qualify Polyspace code verifiers and our other critical tools, we created individual tool qualification documents that included a detailed validation plan. The plan comprised three core areas: correct tool function, structured tool development, and direct experience of tool use.

For the first area, correct tool function, we used test cases, procedures, expected results, and other qualification artifacts from DO Qualification Kit and IEC Certification Kit (Figure 2). For the second area, we also reviewed the certification authority TÜV SÜD's assessment of the structured development process used to create Polyspace code verifiers. For the third area, we documented our own use of Polyspace code verifiers and combined this with additional Polyspace product usage information so that we could adapt the certification kit based on our own use cases.


Certification Artifacts Explorer
File Edit Help
Certification Packages
IEC 61508 / ISO 26262
IEC Certification Kit (R2011a)
Certification Artifacts
ecoder
r2011a
baseline
doc
outputs
tests
certkitiec_ecoder_cdt.pdf
certkitiec_ecoder_cdt.rtf
certkitiec_ecoder_certificate.pdf
certkitiec_ecoder_certreport.pdf
certkitiec_ecoder_modelList.m
certkitiec_ecoder_tests.m
certkitiec_ecoder_tqp.pdf
certkitiec_ecoder_tqp.rtf
certkitiec_ecoder_workflow.pdf
plccoder
sldv
slvnv
polyspace
Certificate
Preview:
Find
55%
ЕРТИФИКАТ ◆ CERTIFICADO ◆ CERTIFICAT
TÜV SÜD
Help
Apply

Click to open C:\Program Files\MATLAB\R2011a\toolbox\qualkits\iec\ecoder\r2011a\certkitiec_ecoder_cer

CERTIFICATE
No. Z10 11 01 67052 005

Holder of Certificate: The MathWorks, Inc.
3 Apple Hill Drive
Natick MA 01760-2098
USA

Factory(ies): 67052

Certification Mark:

Product: Software Tool for Safety Related Development

Model(s): Embedded Coder™

Figure 2. The Certification Artifacts Explorer of the IEC Certification Kit, showing the TÜV SÜD certificate for Polyspace products.

All together, we had more than 10 separate measures for Polyspace qualification, and all measures passed without deviation, enabling us to demonstrate to customers and regulatory agencies that our tool strategy and processes were sufficient to validate Polyspace Client for C/C++ and Polyspace Server™ for C/C++ as verification tools.

Polyspace Code Verification in Day-to-Day Development

In developing the diesel engine control software, MTU developers used Polyspace code verifiers to check their code for run-time errors before checking it into the version control system. This level of informal testing gave developers immediate feedback on their code, enabling them to address any outstanding issues before formal integration testing. MTU build engineers also ran the Polyspace code verifiers as part of the automated nightly build and test process, using the results to identify areas of code that needed more developer attention.

Developers were not permitted to submit code with known run-time errors (highlighted in red), but they could submit code with unproven (orange) or unreachable (gray) elements. Each of these elements, however, had to be justified, which meant explaining why it was not an issue. For example, as a defensive programming practice, the developers implemented each `switch` statement in C with a `default` option that could not be reached via normal operation. These `default` options were correctly highlighted in gray. Each instance was deemed justified because we knew exactly what was causing the Polyspace code verifier to mark it as unreachable code.

Polyspace code verifiers provide access to information that explains why every code element marked as orange is deemed unproven. For example, they may highlight the use of absolute memory addresses, which are sometimes hard-coded in embedded software. In other cases, they may note that the operation could result in an overflow or underflow condition. The development team is then responsible for justifying this potential failure condition or correcting the code as needed.

Formal Verification

After integrating all the code into the version control system, we ran Polyspace code verifiers to recheck the entire code base. A formal review team checked and justified each code element marked as red, orange, or gray. Embedded control software typically contains infinite loops. Polyspace products correctly identified these loops as nonterminating, but no other code as red.

Because this was the first time we had used Polyspace products on an IEC 60880–governed project, we made the decision early on to conduct a manual review in parallel with the Polyspace verification. This manual review required a lot of effort, and did not uncover a single additional issue. A major drawback of manual reviews is that they are dependent on the human reviewers, and therefore are not repeatable. Polyspace code verifiers, in contrast, deliver consistent results no matter how many times they are run on the same code. When we've fixed a problem, we can see that it has been resolved because the corresponding code is colored green, and we can ensure that we have not inadvertently introduced new problems.

Expanding the Use of Polyspace Code Verification

With the diesel engine control software in the final stages of the approval process, MTU engineers have begun to employ Polyspace code verifiers on other projects, including several commercial non-safety control software systems built using Model-Based Design. On these projects, we first model the system with Simulink® and Stateflow® and then generate C code using Embedded Coder™. We use Polyspace products to verify the generated code. Each issue that is identified in the source code is linked back to the Simulink model, enabling us to trace potential problems to their source.

Products Used

- Simulink
- DO Qualification Kit
- Embedded Coder
- IEC Certification Kit
- Polyspace Client for C/C++
- Polyspace Server for C/C++
- Stateflow

Learn More

- Verifying Code When Software Reliability Is Critical
- Embedded Software Verification for IEC 61508 and ISO 26262

See more articles and subscribe at mathworks.com/newsletters.

Published 2012
92013v00