

# Removing Run-Time Errors from AUTOSAR Components Using Polyspace Code Verifiers

By Alexander Much, Elektrobit

Send e-mail to [Jay Abraham](mailto:jay.abraham@elektrobit.com)

A car built today contains 100 to 1,000 times more software than one built 10 years ago. To help organizations manage the growing complexity of automotive software, automobile manufacturers, suppliers, and tool developers collaborated to develop AUTOSAR (AUTomotive Open System ARchitecture). Elektrobit's EB tresos AutoCore is an industry-leading implementation of AUTOSAR-compliant software for automotive ECUs. To ensure vehicle safety, we must verify that these AUTOSAR components are free of overflow, divide-by-zero, out-of-bounds array access, and other run-time errors.

EB tresos AutoCore has thousands of possible software configurations that can be deployed on any hardware currently used in the automotive market. Using bounds-checking software to identify run-time errors in such an application is insufficient because bounds-checkers can only detect errors triggered during a test, and an unworkable number of tests would be required. Instead, we use Polyspace® code verifiers. Polyspace code verifiers identify run-time errors without requiring code execution, which means that we do not need to implement tests specifically designed to capture them.

In the first two months of use, Polyspace code verifiers identified the 97% of our code that was free of certain run-time errors, enabling us to focus our code reviews and further tests on the remaining 3%.

## Challenges of Verifying Highly Configurable Software

In many ways, EB tresos AutoCore is the successor to OSEK, the real-time operating system installed in millions of ECUs every year. As such, EB tresos AutoCore needs to be at least as reliable as OSEK. The challenge is that EB tresos AutoCore is much more sophisticated software: In addition to supporting OSEK classes, it must comply with AUTOSAR standards and support all AUTOSAR classes (Figure 1).

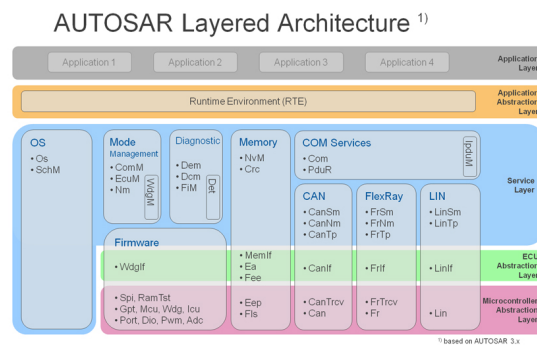


Figure 1. AUTOSAR architecture. Virtually all AUTOSAR modules are covered by EB tresos AutoCore.

The base software comprises approximately 100,000 lines of code. In addition, EB tresos AutoCore is highly configurable, with more than 15,000 parameters, many of which have a range of values.

While identifying code with the potential to cause run-time errors in software with so many configuration parameters can be a tedious and difficult process, for the safety of the overall system it is vital. Dereferencing an invalid pointer, for example, can cause an ECU to halt, resulting in sudden acceleration, deceleration, or other unexpected behavior.

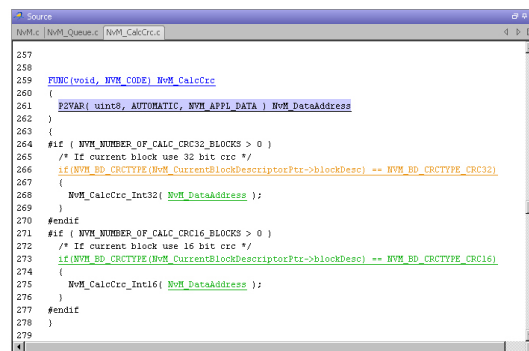
In the past, we tested for run-time errors by instrumenting the code and conducting module tests in which the embedded software processed a few thousand predetermined FlexRay or CAN packets. Even after thousands of tests with bounds-checking software repeated on dozens of hardware architectures, we could not guarantee that all errors had been identified because we had no way to ensure that the test packets we sent would trigger all potential run-time errors in the code. In addition, too much instrumentation can slow the processing, causing the software to miss packets arriving in real time.

Polyspace code verifiers do not require code instrumentation, and our tests check for run-time errors exhaustively, not just in specific scenarios. For each instruction, the Polyspace code verifier analyzes all possible values of every EB tresos AutoCore variable and parameter.

## Compliance with ISO 26262 and Other Standards

Ensuring compliance with ISO 26262 functional safety standard for road vehicles involves demonstrating that the design fulfills the safety requirements, that the architecture accurately reflects the design, and that the architecture is implemented correctly. Polyspace code verifiers help us with the often time-consuming third phase.

All safety-related functionality in EB tresos AutoCore is consolidated into modules that provide, for example, memory protection or end-to-end protection for messages sent via the bus. The code in each module requires thorough review to identify implementation problems. Polyspace code verifiers save us a great deal of review time by highlighting in green the lines of code that have no run-time errors and in orange the lines that may contain errors (Figure 2). One EB tresos AutoCore module has a single orange line of code, and all the rest is green. As a result, we know exactly where to focus our efforts.



```
257
258
259 FUNC(void, NVM_CODE) NVM_CalcCrc
260 (
261     PVAR( uint32, AUTOMATIC, NVM_APP1_DATA ) NVM_DataAddress
262 )
263 {
264     #if ( NVM_NUMBER_OF_CALC_CRC32_BLOCKS > 0 )
265     /* If current block use 32 bit crc */
266     if(NVM_BD_CRCTYPE(NVM_CurrentBlockDescriptorPtr->blockDesc) == NVM_BD_CRCTYPE_CRC32)
267     {
268         NVM_CalcCrc_Int32( NVM_DataAddress );
269     }
270 #endif
271 #if ( NVM_NUMBER_OF_CALC_CRC16_BLOCKS > 0 )
272     /* If current block use 16 bit crc */
273     if(NVM_BD_CRCTYPE(NVM_CurrentBlockDescriptorPtr->blockDesc) == NVM_BD_CRCTYPE_CRC16)
274     {
275         NVM_CalcCrc_Int16( NVM_DataAddress );
276     }
277 #endif
278 }
279
```

Figure 2. Code sample showing lines that may contain run-time errors and lines without run-time errors, highlighted in orange and green, respectively.

In addition to complying with AUTOSAR and ISO 26262 standards, our code must comply with MISRA C<sup>®</sup>. Polyspace code verifiers detect MISRA C coding violations, enabling us to minimize the number of tools we use for static and dynamic code verification.

## Improving Performance Through Shared Memory Checking

Some configurations of EB tresos AutoCore contain shared global variables. Accessing these variables concurrently from multiple processing threads can cause unpredictable ECU behavior. The standard solution to this problem is to implement critical sections and locks on the variables so that only one thread can access them at a time. While this is the safest approach, it is not always the most efficient. Overuse of critical sections can degrade system performance because interrupts must be temporarily disabled as each critical section is processed.

To avoid slowing the system, we must identify which shared variables require critical sections and locks. This is not particularly difficult in a small module, but it becomes complicated in a complete system. We use Polyspace code verifiers to perform this analysis and determine whether locks are properly configured. After conducting this analysis on EB tresos AutoCore code, we found and removed bottlenecks that were hindering system performance.

With EB tresos AutoCore now in production, the number of active AUTOSAR projects at Elektrobit is growing rapidly. Among them is a project to deliver driver assistance systems to a major European automotive manufacturer.

Polyspace code verifiers enable us to demonstrate conclusively that the software we deliver is free of certain run-time errors. More importantly, they enable us to do so faster, more thoroughly, and with less manual review than was previously possible.

#### Products Used

- [Polyspace Client™](#) for C/C++
- [Polyspace Server™](#) for C/C++

#### Learn More

- [Polyspace Embedded Software Verification](#)
- [Developing AUTOSAR Components with MathWorks Products](#)

See more articles and subscribe at [mathworks.com/newsletters](http://mathworks.com/newsletters).