embeff
BETTER EMBEDDED.

White paper: Risks in Off-Target Unit Testing

A comparison of on-target and off-target execution

Usually, embedded firmware is developed on a desktop computer.

Code that is not directly dependent on hardware can be compiled with a conventional desktop compiler, and executed on the development computer (off target). This off-target approach is often chosen in order to avoid long compile-flash-debug loops in the microcontroller.

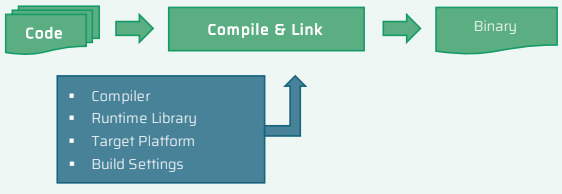
However, it should be taken into account that in certain circumstances, the code tested in this way might behave differently when executed on the real target system (on target). In the most favorable case, an error could already occur during compilation with the cross compiler. If you are less lucky, the compiled code will behave differently in some aspects. You could find that code that has been "verified" in the off-target process is suddenly no longer correct.

The off-target approach is often chosen for the execution of unit tests. This is very useful from the point of view of efficient development. However, if these tests are not also executed on target – with potentially different results! – a purely "perceived security" could quickly result.

This white paper demonstrates the key technical reasons for differences between the on-target and off-target approaches.

Differences in generation

Compiler and Runtime Library



Different compilers provide different support for features. This becomes particularly problematic if language features from comparatively recent standards (for example C17 and C++20) are used. The most commonly-used desktop compilers (Visual Studio, gcc, clang) almost always support significantly more features than a cross-compiler of the same age. Code that utilizes such features will not compile for the target platform, so it must be painstakingly rewritten.

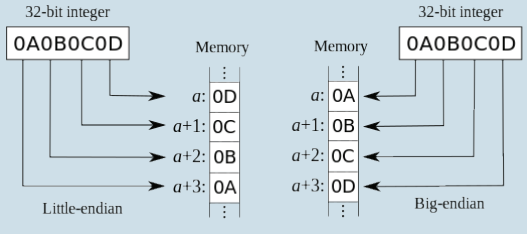
Compiler and runtime libraries have bugs. This may seem surprising at first, but it is a natural consequence of the complexity of these tools, which has become enormous. The actual extent of this potential source of error is dramatic.

An actual bug in the GNU-Embedded Toolchain by ARM illustrates the effects: Bugticket #1527413 "4.9 series reproducibly corrupts Register R7".

Under very special circumstances, a processor register can be changed unintentionally. The sample code for reproduction would successfully run unchanged, using a different compiler. In order to ensure that an error of this type can be detected, the code must be executed on target.

Target Platform

The size of data types in C and C++ is implementation-specific. For example, a pointer on the development PC today is 8 bytes. On a 32-bit target system, however, pointers are usually only 4 bytes. The same applies to long data types. Furthermore, there is the so-called alignment. Processors sometimes need a memory address that is aligned to the size of the word when reading/writing a word. The compiler recognizes these requirements and arranges the values accordingly.



The size and alignment of data types results in different sizes and representations of the structure or class of objects in the memory. Words are also interpreted differently, as little or big endians.

Particularly in low-level embedded code, memory is often interpreted, copied and transferred byte by byte. The code quickly and unwittingly becomes dependent on the target platform, due to the displayed properties.

Differences in generation

Build Settings and Optimization

Language standards, such as C and C++, include undefined behavior.

If a program contains code that is not defined by the standard, the compiler can treat it as arbitrary. Compilers make use of this permissible option, especially with regard to optimization.

```
int f(bool parameter) {
      int a;    // uninitialized local variable
      if(parameter) { a = 42; }
      return a; // potential access to uninitialized variable
}
```

The code shown is undefined, because access to an uninitialized variable could potentially be executed. The effects are analyzed by means of the assembly output of an ARM gcc 8 compiler:

- While optimization is switched off, the parameter is checked for inequality with 0, and only then the code returns 42. Otherwise, the value of the local variable “a ” is returned. Memory has been reserved on the stack for this purpose, but it has not been initialized. Therefore the return value depends on the contents previously stored in this RAM location.
- If optimization is switched on, the parameter value is not evaluated, and the code always returns 42. The existing undefined behavior allows the compiler to execute this drastic – and not very intuitive! – shortcut.

The optimization level can therefore have a considerable influence on the results of a program. The Compile and Link settings must be observed during the verification process.

Differences in Execution

In the previous section, we demonstrated the differences in the on-target and off-target approaches, that already exist for generating a binary. This section briefly highlights the differences when executing the binary.

Runtime Environment

The microcontroller's runtime environment differs from that of the desktop PC in many ways. Some properties are obvious, for example the limited RAM and therefore also the stack on the target system. But some other aspects only become clear upon closer inspection. If a heap is available on the target system, it will have different internal characteristics (allocation strategy and fragmentation).

If you are using a real-time operating system, it may offer a port that is compatible with the desktop PC. However, all operating system calls behind the common API will still have significantly different types of implementation.

Hardware

In the case of an off-target approach, the binary is executed via the hardware of a desktop computer. Of course, the processing CPU differs substantially from the target platform's microcontroller.

As we have already shown for compilers, microcontrollers can also contain errors. There is an erratum for almost every MCU, that lists such errors. In most cases, however, they involve errors in the peripherals of the microcontroller. Particularly complex peripheral circuits, such as DMA controllers, are affected. In principle, proper execution regarding such errors can only be done on target, since both the peripherals and the associated code are platform-specific.

Although much less common, the computing core of a microcontroller is also subject to structural errors. One example is the problem with the ARM Cortex M4F: FPU Errata 1299509 **"Fused MAC instructions give incorrect results for rare data combinations ".**

If there is a simulator for the target platform, it can be used for the execution. However, simulators themselves can contain other errors, or they might fail to correctly reproduce the errors present on the real hardware.

Summary

Developers execute code on the development computer, in order to shorten the long feedback loops (compile-flash-debug) associated with on-target execution. This white paper has highlighted the main points of difference between the structure of on-target and off-target execution. Various influencing factors can result in "verified" code on the target system suddenly no longer working correctly in the off-target process.

A thorough understanding of these effects is essential, in order to avoid errors. Nevertheless, in practice, a verification on the target system should definitely be carried out, at least periodically.

The embeff ExecutionPlatform offers an easy way to integrate on-target testing over all your microcontroller types and families. It supports all major unit test frameworks.

embeff GmbH
Willy-Brandt-Allee 31 b · D-23554 Lübeck · Germany
Tel. +49 451 16088690 · Fax +49 451 16088699
info@embeff.com · www.embeff.com

embeff
BETTER EMBEDDED.