C Compiler Validation for Embedded Targets

Qualifying Compilers for Use in Safety-Critical Projects

By Olwen Morgan

1. Introduction

These notes have been prepared for those seeking validation services for C cross-compilers for embedded targets. Some may find validation unfamiliar and the aim of this document is to explain the essentials so that it is clear what to expect from our compiler validation services.

2. What is compiler validation?

Compiler validation is simply highly controlled, repeatable and reproducible testing of a compiler using a recognized set of test programs commonly known as a *validation suite*. The purpose of such testing is to provide a reliable indication of how well a compiler complies with the standard for the language that it implements.

3. Who needs compiler validation?

Typically compiler validation is sought in connection with the development of safety-critical, security-critical or mission-critical systems. Until recently, compiler validation has been recommended but rarely formally required for the development of safety-related software complying with safety standards such as IEC 61508 Part 3, ISO 26262 Part 8 Section 11, and/or comparable provisions in similar industry-specific standards. Most validations to date have been for large-scale system developments. This is beginning to change as a result of technical advances in microcontroller design.

For example, various microcontroller manufacturers now offer safety-rated microcontrollers. An increasingly popular form is the dual-core lockstep microcontroller for which regulatory bodies are beginning to issue safety certifications covering the hardware. These microcontrollers offer very high safe failure fractions through their physical and logical design. One weakness, however, is that both cores run the same object-code image. For software to match the safe failure fraction achieved by the hardware, it is now becoming more important for the critical parts of development tool chains to be proved fit for purpose.

To date most compilers used in embedded system development have been accepted for use based on their history of reliable operation. Developments in hardware safety are causing a shift towards the use of formally validated compilers where previously validation would not have been required. Clients can now expect regulatory bodies to require compiler validation more and more often for critical system developments. It is expected that this will be seen initially in projects based on new safety-rated microcontroller designs.

4. Why the compiler developer's testing may not be enough

Most compiler suppliers make extensive use of recognized test suites, such as SuperTest™, in their product development. This however, can really only establish that a compiler is *qualified* for a of limited range of critical uses. It does not establish that a compiler is fit for purpose in any specific use. For C compilers used in (safety-)critical projects it is important to understand why this is.

The C language standard makes clear that an *implementation* is a specific configuration of hardware platform and compiler software used under particular compiler invocation options. However thoroughly a compiler supplier may test his compiler, it is likely that:

a) the testing will have been done *on-host* (using a simulator or emulator rather than real target hardware) because *on-target* testing may not be flexible enough or can take too much time,
b) testing has not been done with exactly the same compiler options that are used in a specific embedded development project.

Consequently the compiler developer's testing is not actually testing the same *implementation* that the embedded developer will be using. This is not just an issue of definition: in practice, a small change in the compiler options can have a huge impact on the generated object code. Compiler validation services fill this gap by performing tests under the implementation options used for a particular development project and as far as possible on real target hardware.

5. What does a validator do?

By "validator" we mean an engineer who performs validation testing. The validator's task includes the following:

a) setting up and configuring a test host (usually a PC) to a known and repeatable initial state under a specified version of the host operating system,
b) installing a specified version of the compiler to be tested,
c) installing suitable test driver software,
d) preliminary testing of the above to ensure that the programs of the test suite can be satisfactorily run and their results are correctly gathered by the test driver software,
e) running the tests on-host to provide assurance of the integrity of the host platform configuration,
f) running the tests on a specified externally connected target (usually in the form of a development board connected via a debugging adaptor),
g) analyzing any test failures to determine their causes (together with any test re-runs required),
h) preservation of the test transcript produced by the test tool,
i) preparation and issue of a test report,
j) issue of a test certificate.

These steps are conceptually very simple but the key constraints are that they must be both repeatable and reproducible. To be repeatable it must be possible for the same tester using the same test system to get the same results on successive test runs. To be reproducible it must be possible for a different tester using a different but technically compatible test system to get the same results on separate test runs. In practice this means that testing *must be controlled to the same standards that are applied to certification testing laboratories*. Testers must be specially educated in order to be qualified to undertake testing to this degree of rigour. Most systems engineers, among them even very experienced ones, are not actually qualified to this level and most in-house test systems seen by the author would not be fit-for-purpose in compiler validation.

6. On-host and on-target validation times

On modern PC hardware a test suite containing millions of tests can usually be run on-host several times in a day. On-target testing takes considerably longer because the total elapsed time is dominated by the time required to upload tests to the target environment and download the results from that environment. Actual test execution times are small in comparison. A large test suite may take several days to run on-target, and maybe even weeks if the turn-around times are long.

An evolving practice in performing validations for clients is that a first validation is conducted on the host, then a trial validation is conducted on the required target simply to find out how long it will actually take to run. Clients for validation services should therefore be prepared for providers of such services to take a little while before giving definite elapsed time estimates for validation work. For the moment it is the validation provider's responsibility to ensure that:

a) validation clients understand that for a previously untested compiler/target combination, initial estimates will be subject to uncertainty,

b) validators are themselves prepared to undertake investigative work to resolve any time uncertainties before quoting specific terms and costs to clients.

It is clearly in the best interests of both clients and providers that they make allowance for initial time uncertainties until experience of running a large test suite has accumulated to provide a better basis for estimation.

7. Front-end and back-end testing

Different test suites test different aspects of a compiler's behaviour. Basic conformance suites test that the compiler observes the standard syntax of the language and can correctly compile at least one instance of each language construct. Tests that do this are very good exercisers of a compiler's front end. Among contemporary C compilers, it is now relatively hard to find serious errors in their front ends.

By comparison the back-ends (optimizing transformations and code generators) of compilers are less well tested. This is particularly true of C compilers where many compiler options bear on technical matters at the object-code level. To exercise the back-end needs either a large test suite (such as SuperTest) whose design is based on a boundary-value test coverage strategy, or a set of tests generated by a random stress test generation tool. In practice, and in aid of repeatability and reproducibility, the most controllable approach is the use of a large fixed test suite specifically designed to cover an appropriately strong domain of boundary-value test cases. The strength of such test suites is that they provide a strenuous exercise of a code generator in how it handles arithmetic, which is generally of more concern to embedded system developers than front-end compliance.

8. Validation reports and transcripts

Validation reports must contain at least the following elements:

a) a technical description of the steps taken to configure the test system and run the tests,
b) resolution of any test failures,
c) transcripts of all test runs as produced by the test driver software, a certificate signed by the tester confirming that the tests have been done as described and that the test transcripts are authentic.

For ease of presentation and preservation, test transcripts are most conveniently presented on (non-modifiable) recorded optical media with all other elements of the test report as printed items.

It is normal practice for testers to retain machine-readable copies of results for reference purposes subject to appropriate non-disclosure agreements (NDAs). Currently this author also recommends to clients that such NDAs should be made available to compiler and test suite developers as this is likely to be of help to them in eliminating any bugs found by on-target testing. A further convention is that the validation provider retains all IPR in test reports and transcripts, thus giving it the legal right to require that any copies presented for regulatory approval are as produced by the tester and do not have any parts removed by the client. (Unfortunately this has been known to happen.)

9. Other supporting services

Depending on the scope of service offered, validation providers may offer further advice in dealing with regulatory authorities regarding the documentation of compiler validation procedures in safety cases.

It may happen that errors are found during testing for which the developer needs to take mitigating actions during project development. A qualified validator should have the expertise to advise on such mitigations and how they can be tested. Robustness of a mitigation can be demonstrated by re-running relevant failing test programs from the test suite and then running the programs that test the mitigation. When this is done under laboratory conditions, the advising validator must not participate in any retests where he/she has had a role in specifying what the mitigation tests should be. Ideally this should also be avoided in testing that is not subject to laboratory-level certification procedures. For the moment, however, a pragmatic approach is acceptable provided that the validator's role in post validation advice is subject to agreement between the client and the validation provider.

10. Conclusion

The author and Solid Sands hope that (prospective) validation clients find these notes helpful and we will be glad to answer any specific queries that may arise.