Repeatability and Reproducibility in C Compiler Testing:

Why testers sweat the details

First Edition
by
Mrs Olwen Morgan CITP, MBCS
and
Eur Ing Chris Hills BSc , C. Eng.,
MIET, MBCS, FRGS, FRSA

The Art in Embedded Systems comes through Engineering discipline.

Contents

Repeatability and Reproducibility in C Compiler Testing

Why testers sweat the details

Organisations validating systems against safety standards are increasingly demanding evidence that the compiler has been validated/tested. Well that is not a problem - you just buy a test suite and run it don't you? Well, no, you don't! There is a lot more to it than that and the details really matter. Get one "minor" detail wrong and the whole thing could be invalid. This is: ***Why compiler testers sweat the details***

1 Introduction

So you have your compiler and a nice new test suite: surely any half decent programmer can run the test suite to validate a compiler? Actually that is like saying I have some bricks, doors and windows in a pile therefore I have a house that will pass inspection. To start with, not all test suites are created equal.

A VEB Trabant, a Rolls Royce Silver Shadow and Bugatti Veyron are all motorcars. One has very different build quality to the other two and the two high quality vehicles are aimed at very different markets. The choice of compiler test suite will be covered in another paper, as it requires an in-depth and detailed discussion. For this paper we will assume that by some good fortune you have the appropriate test suite.

It's not just the big things like "Which compiler test suite?" Compiler validators will be very fussy about the technical details of compiler validation and clients for validation services may sometimes not understand why the tester is paying such close attention to what may seem like minor configuration matters. The purpose of this paper is to explain why the tester has to be so careful so that you will also understand why not just anyone can do compiler testing and why compiler testers sweat the details.

2 Repeatability and reproducibility

In any form of testing or measurement it is important to ensure that results are repeatable and reproducible.

2.1 Repeatable

A testing process is repeatable if, and only if, repeated testing of the same property by the same tester using the same test equipment produces results that satisfy defined criteria of agreement. That is: every time you

run the tests on the same compiler build you should get exactly the same results on that test rig.

2.2 Reproducible

A testing process is reproducible if, and only if, repeated testing of the same property by diverse testers using different (but typically technically compatible) test equipment produces results that satisfy defined criteria of agreement. Given the same criteria of agreement, reproducibility is a stronger requirement than repeatability.

That is - if some one else uses a different test system, set-up to an equivalent but not identical specification, and runs exactly the same tests they should get exactly the same results.

Nobody would have any confidence in a testing process that was not repeatable or not reproducible. These are the cornerstones of accurate scientific testing and measurement. However achieving repeatability and reproducibility is not as simple as it sounds. There are many seemingly trivial or inconsequential details that can upset either repeatability or reproducibility.

3 Things that can go wrong for repeatability

In compiler testing many things can undermine repeatability and, ipso facto, reproducibility. The design of and adherence to measures to ensure repeatability constitutes the foundation of high-integrity compiler testing. Potential problems include, but are not limited to:

- Incorrect identification of the compiler under test
- Incorrect identification of compiler invocation options
- Unstable test platform configuration
- Inadequate test isolation
- Lack of robustness in test driver software
- Lack of integrity in test reporting software

Lack of proper attention to any of these matters can render results unrepeatable, even when repeating the same tests on the same test platform twice in succession under the control of a test automation script. Obviously the compiler tester must take steps to ensure that this kind of thing cannot happen. These steps lead to very strict procedures for test set-up and execution.

3.1 Identifying the compiler under test

Sometimes people will speak of, for example, "the GCC compiler version 4.3" thinking that this uniquely identifies the compiler for testing purposes. It doesn't. However to say "the IAR Cortex compiler, version 7.123" will uniquely identify the compiler, as it is a specific build and everyone who has the IAR Cortex compiler 7.123 has exactly the same package. This is because the IAR compiler is issued as a binary from a single point and from a strictly controlled development process. This will be true for other commercial compilers such as those from Keil, Green Hills, Byte Craft and, dare I say it, Microsoft compilers. So when IAR/Keil/Green Hills/MS etc. say they have 10,000 users for "proven in use" that means that there are 10,000 users using an identical compiler [binary] package. However you will still need to confirm the details of the contents of the rest of the compiler package.

The GCC (GNU Compiler Collection) is a collection of packages from many sources that may differ depending on which of the various distributions you pick. While the GCC compiler front end may be reasonably stable across most GCC compilers, the back end and associated components may differ. This may include the situation where a set of identical sources is built by different (or rather, non-identical) compilers. Your GCC compiler, despite millions of GCC users globally, will have a "proven in use" population of one - you.

The compiler tester needs to identify all the components of the compiler suite - in detail. The required details include, at least the following:

- The compiler developer's version number for the compiler and who the developer is. For GCC, you also need the developer and the distribution and patch levels of all the components as these come from multiple sources.
- The version numbers, patch levels and patch histories for any associated libraries, linkers, other components, target simulators and on-target debugging protocol software or devices.
- When testing on-target using a development board, the version of the board, including silicon revision level, of the target microcontroller. Some

MCUs have multiple revision levels, which are usually indicated by additional suffixes on the MCU part number (although these are not normally part of the part's order number). There have been cases where a "feature adjustment", a bug fix between revisions, or even a mid-run change of the MCU reel on a pick and place machine has affected the teat results.

Failure to accurately identify and record these items can render the results of even a single test program wholly unrepeatable.

3.2 Specifying the invocation options

Errors in specifying compiler invocation options are, in the authors' experience, the second most common cause of repeatability problems. Most compiler vendors delight in telling you how many MCU variants their compilers support, especially since the spread of Cortex. Then there can be memory models (for those programming MCU's like the 8051, which itself has nearly 1000 variants) and switches for compiling for size or speed. There may be library options that use different standard libraries that behave differently e.g variants of printf . The sizes of the integral types may affect tests. What size are char, short, in and long? This is apart from looking at all the implementation-defined, unspecified and undefined parts of the C standard (there are some 20 pages of these in C99 & C 11) to check what the compiler does with them. Is plain char signed or unsigned in your compiler?

Here are a couple of compiler invocation strings for different compilers; the compiler validator will need to know precisely what every item is, and what it does.

Compiler A

```
--c99 -c --cpu Cortex-M3 -D__
  EVAL -D__MICROLIB -g -O3
  --apcs=interwork --apcs /ropi/
  rwpi --split_ldm --split_sections
  --strict --enum_is_int --signed_
  chars -DSTM32F10X_MD -o ".\Obj\*.o"
  --omf_browse ".\Obj\*.crf" --depend
  ".\Obj\*.d"
```

Compiler B

```
HOLD(128,0,0) OPTIMIZE (7,SPEED)
   BROWSE ORDER NOAREGS DEBUG
   OBJECTEXTEND CODE LISTINCLUDE
   SYMBOLS TABS (2)
```

The options need to be recorded in a single file. This is used as the source for all options when the compiler is invoked under test. The compiler validator needs to understand what all the invocation options do and how they interact.

There are also going to be default options that are not shown on the command line. While it is not always straightforward to identify what the default options actually are, or how they interact, it is essential to state them explicitly: they may have been overriden by other options.

3.3 Unstable test platform configuration or environment

For critical systems it may be a requirement that the test system be maintained in commission for the market life of the product. For commercial aircraft, railways, the nuclear industry and others this can exceed 30 years. In 2015 the authors were called on to provide test hardware for a project that had first shipped in 2000: system maintenance was required and therefore there was a need for retesting.

You can't simply do compiler validation by reloading the test suite on a development team PC that is continually changing. One option is to freeze the hardware configuration of the test platform and mothball it in a quarantined store. This also requires retaining the relevant operating system software release media.

To re-establish a mothballed test platform it must be set up from scratch by reinstalling all necessary software configuration items. This can be a surprisingly difficult operation if not done very carefully. It must not rely on software downloads from the internet; if they are still available they may have changed over time. It must also set up the original tester's operating system shell environment from scratch. As far as possible it must not rely on the user's log-in environment variables but be set

up to use testing-specific variables. Special environment set-up tools are often the best option for this.

It is important that the test procedure is designed to ensure that the tester's shell environment is not corrupted during test operation. This is especially important when a failed test can leave the compiler invocation environment in an indeterminate state.

In a short paper such as this is it not possible to identify all of the things that can go wrong in test platform configuration but clients for test services should appreciate that problems in this area are a substantial cause of repeatability problems.

Suffice it to say the compiler tester's computer must be one dedicated for the purpose and not used for anything else as you are going to have to do a complete clean install from the OS upwards. The authors have such a (custom built) PC and sets of external hard drives containing disk images that preserve entire software environments for repeat testing. You are not going to simply run the compiler test suite on a random development PC that changes state every time it is turned on.

3.4 Inadequate test isolation

While a modern compiler should pass almost all compiler validation tests, there remain occasions on which a test will fail. When running tests on-target, it is essential to ensure that the state of the target can be re-set and that the test driver can recover from the failure so that it does not prejudice the results of subsequent tests.

In the early days of compiler validation, test suites were small and could be organised so that a single test program contained a single test. This is impractical for modern large test suites running on embedded targets, where a single test program may contain over a hundred individual tests. The test suite should incorporate measures to ensure that when an individual test fails within a test program, that test can be isolated and re-run on its own: typically special software either extracts the failed test from its test program or suppresses all other tests within the same program. The suite developer has to consider considerable technical detail to do this in the most robust way, to avoid re-running the test under test conditions that are inadvertently different from those of its original occurrence.

A lot of this comes down to the design of the test suite. Please see our paper on choosing a test suite for an in-depth discussion of this.

3.5 Lack of robustness in test driver software

A corollary of the need for manageable test isolation is the need for test driver software to be exceptionally robust. It must remain unaffected by the failure of any test and be able to run subsequent tests under the same conditions as preceding tests. This requirement can be largely met by appropriate driver design but again many configuration checks have to be considered to ensure that the appropriate degree of robustness is actually realised.

In general this requires very high quality code written to very strict guidelines. However, as noted, the design of the drivers also has to be very carefully thought through. This is not a trivial thing to do and requires a thorough understanding of how the test suite works.

3.6 Lack of integrity in test reporting software

Finally, the test reporting software must be as carefully engineered as all other aspects of the test driver software. It is not unknown for test report generator software to generate incorrect reports, for example those from a previous test run with different results. Again, careful configuration checks are needed. Version control is paramount and so is picking up the correct files. Time and version stamping is essential, as is the control framework. Mistakes here that give a pass when there should have been a fail will be costly.

The authors know of one case where due to errors the rest results submitted were not accurate and the Notified Body did a complete audit and overhaul of the whole development and test procedures of a project, and required improvements in the project, before issuing certification.

4 Ensuring reproducibility

All of the potential problem areas for repeatability are similarly potentially problematic for reproducibility. After having performed a validation for a client, it is normal practice for the tester to ensure that the client can reproduce the tests on similar but compatible hardware. The normal way for the tester to address this is to perform tests on his/her own test platform, then reproduce them on the client's platform. In this way the client should always be left with a system which can reproduce tests as often as required. To facilitate this, the authors provide test environment software images on suitable media as part of the service, usually on an external hard drive (an SSD these days) and/or a DVD/CD.

Clients should be prepared to work with the tester so that the initial reproduction can be made on the tester's own host platform. This may involve further technical questions to be answered.

5 Working with the tester

Before undertaking compiler testing, a professional tester will ask a validation client for a significant volume of technical information: information essential to providing an assurance of reproducibility and hence to the success of testing. Usually most of the required information can be summarised through a checklist. Asking the client to complete such a checklist is normal working procedure for the tester.

It is not unusual for clients filling in such checklists to make several attempts to get correct information: frequently the issue is the compiler's invocation options. If the client does not have the answer or is not 100% sure, a "don't know" answer is better than a guess. ***Also the client should never assume but check.*** The authors have often asked questions, and the client has been surprised that the correct answers were not what they had assumed them to be.

The validation client may need to consult the compiler vendor in some cases. It is also normal for a tester to work collaboratively with both the validation client and the compiler developer to ensure that relevant information is correctly established.

Clients should be prepared for a fairly interactive process as the tester seeks to obtain all the necessary information. It is not just the tester being pedantic. A lot of detail is needed to ensure that tests are repeatable otherwise the validation process is an expensive waste of time and money and may result in the product certification being withheld.

6 Conclusion

Compiler testing has many technical pitfalls. It is not something someone who is not trained in compiler validation can do: it needs rigorous technical discipline and no corner cutting. Compiler testers are meticulous and they have to be. Certifying bodies appear to be tightening up on compiler validation and have now started to make specific on-target validation requests.

Once a company has been though a compiler validation for a project, subsequent validation for other projects should be a lot faster and smoother, because the developer will have much of the information and will know how to pull the rest of it together with less pain than the first time.

For a compiler tester to get it right, he needs a substantial amount of technical information from the client and compiler vendor. Conveying all this information takes time and may involve repeated dialogue between the parties concerned. Clients for compiler testing should be aware of this in advance so that it does not come as a shock when the **compiler tester sweats the details!**

Repeatability and Reproducibility in C Compiler Testing: Why testers sweat the details

First edition August 2016

Contact the authors at: info@phaedsys.com

Phaedrus Systems Library

The Phaedrus Systems Library is a collection of useful technical documents on development. This includes project management, requirements management, design methods, integrating tools to IDE's, the use of debuggers, coding tricks and tips. The Library also includes the QuEST series.

Copies of this paper (and subsequent versions) with the associated files, will be available with other members of the Library, at:

http://library.phaedsys.com

PhaedruS SystemS

The Art in Embedded Systems comes through Engineering discipline.