Q&A about SuperTest for compiler users

working in safety critical markets

What is SuperTest?

SuperTest™ is a compiler test and validation suite for C and C++. SuperTest contains a tool to run tests (the test-driver), a report generator, and a large collection of tests. SuperTest can be used to verify that a compiler correctly implements the programming languages C or C++.

How are the C and C++ programming languages defined?

The C and C++ languages are defined by ISO standards with the names ISO/IEC 9899:2011 and ISO/IEC 14882:2014. These standards are updated at regular intervals. For example the popular language version 'C99' is specified in ISO/IEC 9899:1999, an older version of the C standard.
The tests in SuperTest are built and classified according to these standards. Some compilers, such as those based on GCC, provide many extensions to the C language. Such extensions are not part of the ISO defined standards.

I am an application developer that uses an off-the-shelf compiler. Why do I need to test this compiler?

Compilers are extremely complex programs with many options. A compiler error can break an application in unexpected ways that may be hard to debug. Even if the compiler supplier has tested the compiler, the compiler was probably never tested for your particular use case (the combination of environment and compiler options).
For applications in safety critical or security domains, it is important that developers have confidence that the compiler does not introduce application errors for their specific use case.

What do I need to test the correctness of a C or C++ compiler?

The short answer is SuperTest. The long answer is that you need a test-suite that verifies the 'behavior' that is specified by the language standards. To test this behavior, one needs a test-suite, a compiler and a host-machine, target-hardware

or a simulator to run the tests. The test-framework will collect the results of the tests and generate a report.

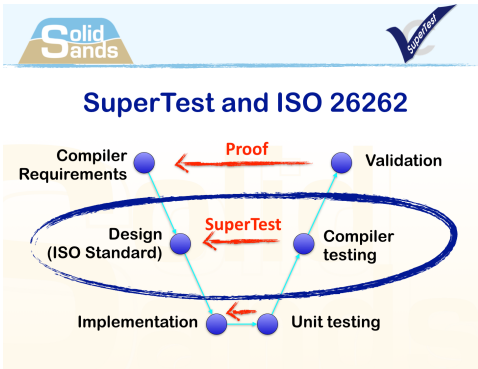
Does SuperTest test all of C and C++, which is necessary for the compiler accreditation?

That is the goal of SuperTest. The tests in SuperTest are organized according to the language standards so it is easy to find the mapping between the tests and the language standard. There are also parts of the language implementation that are 'implementation defined'. In that case, SuperTest tries to cover the different choices as much as possible. A large area of implementation defined behavior in C and C++ is related to the arithmetic sizes of the fundamental types. For that special case, SuperTest includes as many as 33 variants of bit-precise arithmetic tests, build on 33 different arithmetic models. If your required variant is not yet there, Solid Sands will generate a new arithmetic test suite for that model.

What about the standard C library, do you test that?

Yes, the C library definition is part of the language standard. In fact you can combine this with the compiler verification, the configuration can be the same.

I need to qualify my compiler for ISO 26262, what do I do?



Qualification of the compiler may be needed if the compiler is used to generate code for safety critical applications. In such cases, the compiler is a tool and the rules for tool qualification are defined in Section 8.11 of ISO 26262. Different methods can be used for tool qualification, of which testing against the tool specification (in this case the language standard) is one possible technique. SuperTest can be used for this because it is build and organized according to the language standard. Furthermore, tool qualification requires careful documentation of the use case (such as compiler options used), the test setup, compiler version and other relevant parameters. The documentation must be precise enough to repeat the qualification process.
Also, the documentation must define the use case to which the application developers adhere. If compiler failures are found, they must be documented as well, with recommendations or procedures ('mitigations') to make sure that the developers do not run into these known failures.

Has SuperTest been qualified for IEC 61508 or ISO 26262?

SuperTest by itself cannot be qualified for IEC 61508 or ISO 26262 qualification because such qualification can only be done for a specific use case: in combination with a specific compiler and a specific compiler use case. SuperTest has been used in multiple compiler qualification projects, stand-alone and as part of Q-Kits. SuperTest is usable as such because we have clearly documented how the test-cases correspond to the (language-) specification of the compiler.

How does SuperTest test compilers?

There are two kinds of tests, *positive* and *negative*. *Positive* tests only contain correct C programs and hence should be compiled successfully. For *negative* tests a diagnostic is expected, and compilation is expected to fail.
Every test program includes the file *def.h*. This file defines several macros used by a test program.

Example 1, a simple positive test:

```
/*
 * (c) Copyright 2015 by Solid Sands B.V.,
 * Amsterdam, the Netherlands. All rights reserved.
 * Subject to conditions in the RESTRICTIONS file.
 */

#include "def.h"

MAIN{
    CVAL_HEADER("compound literal (sizeof incomplete array)");
    CVAL_VERIFY(sizeof((int []) {1, 2, 3, 4}) == 4 * sizeof (int));
}
```

This test performs a comparison of sizes using the **sizeof()** operator. On the left side of the comparison there is a compound literal, in this case an integer array of unknown size, which' size is determined by the initializer list of four elements. Therefore, the size is that of four integers. On the right side the control value is computer by multiplying the the size of an integer by four.

C99 Standard (ISO/IEC 9899:1999), Section 6.5.2.5, Paragraph 4:
A postfix expression that consists of a parenthesized type name followed by a brace-enclosed list of initializers is a compound literal. It provides an unnamed object whose value is given by the initializer list.

C99 Standard (ISO/IEC 9899:1999), Section 6.5.2.5, Paragraph 5:
If the type name specifies an array of unknown size, the size is determined by the initializer list as specified in 6.7.8, and the type of the compound literal is that of the completed array type.

Example 2, a simple negative test:

```
/*
 * (c) Copyright 2015 by Solid Sands B.V.,
 * Amsterdam, the Netherlands. All rights reserved.
 * Subject to conditions in the RESTRICTIONS file.
 */

int v [2][ ][1] = { {{3}, {2}}, {{1}, {0}} };

#include "def.h"

MAIN{
    CVAL_HEADER("Only 1st dimension may be omitted.");
    CVAL_VERIFY(0);
}
```

Since an array of incomplete type cannot be constructed, arrays with more than one dimension can only have unknown bound in the first dimension. Therefore, this test is expected to fail at compilation.

C99 Standard (ISO/IEC 9899:1999), Section 6.2.5, Paragraph 20:
An array type describes a contiguously allocated nonempty set of objects with a particular member object type, called the element type (since object types do not include incomplete types, an array of incomplete type cannot be constructed).

C99 Standard (ISO/IEC 9899:1999), Section 6.2.5, Paragraph 22:
An array type of unknown size is an incomplete type.

C99 Standard (ISO/IEC 9899:1999), Section 6.7.5.2, Paragraph 4:
If the size is not present, the array type is an incomplete type.

Example 3, a positive optimization test:

```
/*
 * (c) Copyright 2015 by Solid Sands B.V.,
 * Amsterdam, the Netherlands. All rights reserved.
 * Subject to conditions in the RESTRICTIONS file.
 */

#include "def.h"

int test(int j, int k, int recurs){
    int x = 0;
    int result = 0;
    if (recurs)    { test(0, 0, 0); } /* prevent inlining */
    bb2:
    if(j == 0)     { x = 1;       goto bb12; }
    else {
        bb4:
        if(k == 0) {              goto bb13; }
        else       { k = 0;       goto bb12; }
    }
    bb12:
    if(x == 0)     { result = 20; goto bb2; }
    else           { result = 10; goto bb4; }
    bb13:
    return result;
}

MAIN{
    CVAL_HEADER("irreducible flow and constant propagation");
    CVAL_VERIFY(test(1, 1, 0) == 20);
    CVAL_VERIFY(test(0, 1, 0) == 10);
}
```

This test is intended to verify the correctness of compilers when they try to optimize code. This test is specially designed to contain an irreducible loop. This is rare, but compilers must correctly identify such loops before transformations.

What do I need to run SuperTest?

First, one needs to get a license for SuperTest from Solid Sands. SuperTest is commercial software. Second, SuperTest needs to be installed. Third, one needs to configure SuperTest to use the compiler and target machine or simulator. Fourth, run SuperTest.

How do I install SuperTest?

SuperTest is highly portable and can be installed on any flavor of Unix or Linux, including Mac OS, and Windows. On *nix systems, some developer tools (such as a host C compiler and Perl) need to exist in order to install SuperTest. On Windows, SuperTest is easiest to run in a Cygwin environment. The installation of SuperTest takes only a few minutes. SuperTest only installs in the current installation directory and makes no other modifications to the host-system. So, different versions of SuperTest can be installed side-by-side on the same system.

How do I configure SuperTest?

In order to run a test, the SuperTest test-driver must know which tests to run, how to run the compiler and how to run the compiled program. These are all defined in a few user configurable files and scripts. The top level configuration file defines the compiler's name, compilation and linking flags, the list of tests to run, the name of the LOG directory, etc. Then there are separate script for the compilation of tests and the execution of tests. The test-driver requires that compilation and execution can be done from the command-line, and that it gets feedback about the success or failure of the compilation and test execution.

How do I run SuperTest?

Once SuperTest is configured, running is done by passing the configuration to the test-driver. Typically, one runs a collection of tests, but for debugging one can also run one specific test case. SuperTest has an option to run tests in parallel on the current machine or even on a network of machines, in order to speed up the process.

How do I get the test results?

SuperTest keeps a detailed log of every test-run. SuperTest's 'log-report' command can be used to extract an easy to read and comprehensive report of the test results. It can generate both ASCII reports and HTML reports for easy browsing.

How long does it take to run SuperTest?

That depends on the size of the test-list, the speed of the compiler and the time it takes to (up)load and run a test on the target. The tests themselves are typically

short and do not take much time to execute. A C99 validation of a host-native compiler takes less than 10 minutes to run about 5000 test programs.

Can SuperTest run on small targets?

Yes. Many test programs compile to less than a few kilobytes of code and use few run-time resources. Of course, SuperTest also contains some very large tests, precisely to make sure that they also compile and run OK. These can be skipped easily. SuperTest tests usually contain calls to the diagnostic library that generates easily understandable diagnostics in case a test fails. If a target is resource constrained or has no I/O capability, the diagnostic library can also be skipped completely.

Testing your compilers really isn't that difficult

Contact Solid Sands for more information.